uc3m | Universidad Carlos III de Madrid

Doble Grado en Ingeniería Informática y Administración de Empresas
curso académico (2018-2019)

Trabajo Fin de Grado:

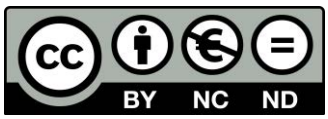
"Diseño de una interfaz gráfica para la conexión y configuración de drones"

Raúl Escabia Martínez

Tutor

Francisco Valera Pintor

Colmenarejo – julio 2019



RESUMEN

Aprovechando la tecnología UDP/IP para el control remoto de los UAV, en este trabajo se desarrolla un software, basado en la arquitectura cliente-servidor, capaz de configurar diversos aspectos de los UAV y automatizar procesos habitualmente hechos a mano desde el terminal. Esta memoria contiene todo el proceso desde el trato con el cliente hasta la instalación del software *ad hoc*.

El software desarrollado cuenta con una interfaz gráfica intuitiva, por lo que ya no se requerirán usuarios expertos para configurar los UAV: se mejora el tiempo malgastado en la configuración y se expande el horizonte tradicional de personal capacitado. Adicionalmente, al considerarse los UAV un sistema crítico, se hace especial hincapié en el tratamiento y recuperación de errores.

Se espera que este software sirva de base para futuras ampliaciones de funcionalidades –sobre el mismo software- o escalados a otros modelos UAV -más allá de SIVA y MILANO- u otros sistemas operativos como Windows, Mac y Android. Todo según las necesidades que vayan surgiendo con los nuevos modelos o los ya presentes.

Palabras clave: UDP, IP, software, UAV, dron, interfaz gráfica, desarrollo de software.

ABSTRACT

Taking advantage of UDP / IP technology for the remote control of unmanned aerial vehicles, in this work we develop a software, based on client-server architecture, capable of configuring the various aspects of unmanned aerial vehicles and automate the processes traditionally done by hand from the terminal. This memory contains the entire process from customer service to the ad hoc software installation.

The developed software has an intuitive graphical interface. Thus we do not require expert users to configure the UAV anymore: time wasted on configuration is improved and the horizon of trained personnel is expanded. Additionally, being UAV considered as a critical system, we lay emphasis on error treatment and recovery.

This software is expected to serve as a base for future expansions of functions –on the same software- or scaled to other UAV models –beyond SIVA and MILANO- or other operating systems such as Windows, Mac or Android. All according to the needs that arise from the new models or the ones already present.

Key words: UDP, IP, software, UAV, drone, graphic interface, software development.

CONTENIDO

ÍNDICE DE TABLAS

ÍNDICE DE ILUSTRACIONES

1. INTRODUCCIÓN, MOTIVACIÓN Y OBJETIVOS

1.1 MOTIVACIÓN

El diseño es una de mis disciplinas favoritas en el área de informática. Toda la carrera he volcado mi esfuerzo y creatividad en solucionar los problemas que plantean unas funcionalidades y sus restricciones adversas. Me puedo pasar horas y horas frente a una hoja de papel en blanco con tal de hallar una solución que me parezca adecuada. Cuando me ofrecieron la oportunidad de hacer el TFG sobre diseñar un sistema para drones, una tecnología 'novedosa' y atractiva, no tarde nada de tiempo en contestar afirmativamente; realmente es un trabajo que me apasiona.

En el ámbito de la configuración de los drones siguen surgiendo nuevos softwares y aún no hay una clara unificación. Es por esto que las marcas tienen que crear sus propios programas y arquitecturas. Un dron se compone de bastantes sistemas y continuamente hay que mantenerlos y hacerlos compatibles entre ellos. Debido a esta diferencia surge este TFG para desarrollar un sistema específicamente para el UAV[1] del INTA[2].

Adelantamos que, en este momento, no existe ningún software que ofrezca todas la funcionalidades de las que se requiere para este proyecto (discutiremos los detalles más adelante en el apartado de 2. Estado del arte). Por lo tanto, se debe diseñar desde cero el sistema.

1.2 PLANTEAMIENTO DEL PROBLEMA

El problema actual que se plantea en la configuración del sistema de comunicaciones IP del SIVA[3], es que se configura manualmente (siendo habitual el uso de una terminal y la necesidad de usuarios expertos). Se quiere solucionar que la susodicha configuración deba hacerse de forma manual y restringido solo a usuarios expertos.

La configuración incluye apartados como: a) puertos e IP; b) tabla de enrutamiento; c) estado de un proceso; d) matar un proceso; e) lanzar un proceso; f) vigilancia de un proceso.

La realización de este proyecto disminuirá los tiempos de configuración considerablemente al automatizar parte de las tareas que no requieran de un humano.

1.3 OBJETIVOS

El objetivo de este TFG es: diseñar, desarrollar y desplegar un software *ad-hoc* para que la configuración del sistema de comunicaciones IP del UAV SIVA del INTA (que se diseñó e implementó con la colaboración de la UC3M[4]).

Los objetivos principales son programar un sistema que:

1. Configure la comunicación y los procesos del router del UAV.
2. Permita ser utilizado por usuarios no expertos.
3. No sature ni la memoria ni la CPU del router de comunicaciones UAV.

[1] Siglas en inglés de: Vehículo no tripulado.
[2] Instituto Nacional de Técnica Aeroespacial.
[3] Un modelo de UAV.
[4] Universidad Carlos III de Madrid.

1.4 FASES DEL PROYECTO

Las fases que se deben cumplir en orden secuencial son:

1. **Planteamiento del problema**: se define la necesidad que se debe cubrir con el sistema.
2. **Estado del arte**: se estudia el entorno y los sistemas que ofrecen soluciones acercadas al problema.
3. **Estudio y selección de alternativas**: se plantean posibles diseños y se decide cual es el adecuado para este proyecto.
4. **Diseño de nivel interno**: se diseñan las funcionalidades que debe tener el sistema.
5. **Desarrollo e implementación a nivel interno**: se desarrolla e implementa el diseño del nivel interno.
6. **Diseño de interfaz gráfica**: se diseña la 'apariencia' del programa.
7. **Desarrollo y acoplamiento de interfaz gráfica**: se desarrolla y fusiona la interfaz con el nivel interno.
8. **Evaluación y pruebas**: se realizan pruebas para detectar errores y, en caso que proceda, rediseñar.
9. **Redacción memoria**: redacción de la documentación necesaria.
10. **Publicación**: subir el software a la plataforma Github.

1.5 ESTRUCTURA DE LA MEMORIA

La memoria se estructura en un total de 9 apartados claramente diferenciados por su número y título. La mayoría de apartados coinciden con las fases del proyecto. Asimismo, se ha seguido las pautas expuestas en la rúbrica de evaluación para redactar los apartados: *2. Estado del arte*, *4.2 Justificación de diseño*, *4.5 Marco regulador legal*, y *4.6. Entorno socio-económico*; exceptuando el 4.2 que ese refiere a "justificación de la solución", todos los demás son homónimos con respecto a lo especificado en la rúbrica.

A continuación, se explican, brevemente, qué contendrán cada uno:

1. **Introducción, motivación y objetivos**: un apartado introductorio al trabajo donde reside un resumen del proyecto y de la documentación. Asimismo, se encuentran definiciones de términos que se utilizan a lo largo del documento.
2. **Estado del arte**: se analiza el UAV con el que se va a trabajar y el software actualmente disponible que satisface la necesidad planteada. Además, se detalla el límite legal del sistema y el impacto socio-económico.
3. **Requisitos**: se especifican las funcionalidades y limitaciones del sistema que se pretende desarrollar.
4. **Diseño[5]:** se detalla y justifica la manera en la que se va a desarrollar tanto el nivel interno como la interfaz gráfica. Definiendo, así, como se implementarán las funcionalidades. Adicionalmente se analiza el impacto socio-económico y las limitaciones legales.
5. **Desarrollo**: se explica la fase de implementación del diseño y los problemas que han surgido durante el mismo. Se deciden y justifican rediseños que no se detallaron en el apartado anterior. Y se expone el resultado final del sistema.
6. **Evaluación y pruebas**: se somete a pruebas los dos aspectos importantes del sistema: nivel interno e interfaz gráfica. Y se deduce si el sistema está preparado para su puesta en marcha.
7. **Planificación y recursos**: se detalla la sucesión de acontecimientos y los recursos invertidos en el proyecto.

[5] A pesar de que el diseño de la interfaz gráfica se lleva a cabo después del desarrollo a nivel interno, se decide agrupar los diseños y los desarrollos debido a que es la forma más intuitiva de leerlo. Sin embargo, se recuerda en varias ocasiones la secuencia temporal.

8. **Conclusiones y trabajo futuro**: se resume los puntos más relevantes del trabajo y se da consejo sobre los posibles caminos que puede tomar el proyecto.
9. **Referencias y bibliografía**: documentación utilizada y recursos electrónicos.

1.6 GLOSARIO DE SIGLAS

Tabla 1 - Glosario de siglas

Abreviación	Equivalente
SO/SSOO	Sistema(s) operativo(s)
TFG	Trabajado fin de grado
IDE	*Integrated Development Environment* (entorno de desarrollo interactivo)
GUI	*Graphical User Interface* (interfaz gráfica de usuario)
UAV	*Unmanned Aerial Vehicle* (vehículo aéreo no tripulado)

1.7 GLOSARIO DE TÉRMINOS

Tabla 2 - Glosario de términos

Concepto	Definición
Ad-hoc	Cuando habalmos de un sistema/software ad-hoc es que se ha creado 'específicamente' para un entorno.
Watcher	Proceso que vigila otros procesos. Habitualmente se ocupa de mantenerlos corriendo.
GCS	*Ground Control System*: nombre que se le asigna a la *Máquina Router de tierra*
IDAN	Nombre arbitrario que se ha asignado a la *Máquina Router UAV*
ssh	Protocolo por el cual una máquina se conecta a otra.
Escuchar	Término informático que se refiere a un proceso que se mantiene a la espera de un evento.

2 ESTADO DEL ARTE

En este apartado se investiga sobre el estado actual de lo relacionado a este TFG. Con el fin de estudiar el software que mejor se adapte a los objetivos propuestos.

Empezaremos detallando lo que es un UAV para contextualizar el sistema que se diseña. Seguidamente estudiaremos otros softwares que satisfacen algunos de los objetivos propuestos; concluiremos cual es la mejor aproximación.

2.1 UAV

El UAV es un vehículo aéreo no tripulado; conocidos actualmente como drones. Aunque su utilidad ha sido típicamente militar, hoy en día sirven para multitud de funciones fuera de ese ámbito (agricultura, vigilancia, investigación…). En concreto, nos centraremos en los modelos SIVA y MILANO; ellos, potencialmente, utilizarán el software que se desarrolla en este proyecto.

El control de los UAV (ambos modelos) se realiza desde tierra mediante protocolo IP. Existe un router dentro del UAV y otro en tierra que se conectan mediante este protocolo, siendo así posible su manipulación a distancia.

Observaremos que son drones más grandes que una persona. Por lo que, su coste es bastante elevado y se considerara crítico el que no falle ninguno de los sistemas integrados (incluyendo el software que se desarrolla). Otro aspecto importante es la autonomía del vehículo, de ahí que uno de los objetivos sea un consumo mínimo de recursos.

A continuación, hablamos de las características de cada modelo:

2.1.1 SIVA (Sistema Integrado de Vigilancia Aérea)

SIVA es un UAV de 4 metros de largo creado, principalmente, para funciones de vigilancia, reconocimiento y ajustes de tiro. Con un peso de 235Kg y una carga útil de 25Kg, su autonomía llega a durar 7 horas. Además, incluye un tren de aterrizaje, no siendo necesario un sistema de captura cuando aterriza.

Ilustración 1 - UAV: SIVA

Fuente: http://www.ejercito.mde.es

2.1.2 MILANO

MILANO es un UAV de 8,20 m de largo basado en SIVA. Respecto a SIVA, posee baja detectabilidad frente a radares y una mayor autonomía. Sus funciones, principalmente son de observación, vigilancia y apoyo a unidades de emergencia; similar al otro modelo. Con un peso de 900Kg y una carga útil de 150Kg, su batería llega a durar 20 horas (casi el triple que el SIVA).

Ilustración 2 - UAV: MILANO

Fuente: www.defensa.com

2.2 MÁQUINA ROUTER

El proyecto de crear la *Máquina Router UAV* se llevó a cabo en 2013 por la UC3M. Este Router es el original que se diseñó y embarco en el UAV y voló en multitud de ocasiones. Al finalizar el proyecto, la máquina fue devuelta a la universidad y se replicó para instalarlo, posteriormente, en el UAV final. Por lo tanto, nos encontramos ante la *Máquina Router UAV* original y al que se utilizará para las fases de este proyecto.

Ilustración 3 - Máquina router UAV 1

Es un ordenador basado en arquitectura PC104 (sistema IDAN fabricado por RTD[6]). Rugerizado con un armazón, soporta golpes y condiciones atmosféricas adversas como: presión,

[6] Véase: RTD (2019)

humedad, calor, frío… Además, integra un conector VGA que se utiliza para la fase de **Desarrollo,** pero no para la de **Evaluación y pruebas** ya que en una situación real no se tiene acceso físico al UAV. Posee conectores Ethernet y puertos USB 2.0 que se configurarán para acondicionar el entorno de trabajo para este proyecto.

2.3 SOFTWARE

A continuación, se listan los softwares más relevantes que cubren parte de las necesidades que plantea el problema; llegando ninguno de ellos a satisfacerlas todas a la vez.

2.3.1 Ansible

Ansible[7] es un software gratuito y con lenguaje propio de programación. Se centra en el aprovisionamiento remoto como puede ser: automatización de procesos; instalación de ssoo; manipulación de procesos/aplicaciones; proveedor de api… etc. Un software bastante completo, que, con un buen diseño y con terceros programas se podría llegar a cumplir el objetivo de configuración. Sin embargo, su complejidad y la necesidad de familiaridad con el entorno de programación hace que solo sea utilizable por usuarios expertos. Y, en caso de diseñarlo para automatizar x número de procesos, se pierde versatilidad a la hora de realizar cualquier operación extra que no estuviera prevista.

2.3.2 SSH[8]/Script

Actualmente, la configuración se realiza a cabo mediante la terminal y una conexión ssh. Una posible alternativa sería la ejecución de scripts preprogramados. Sin embargo, la complejidad de lanzar comandos mediante la terminal y lanzar scripts mediante la terminal es prácticamente la misma. Además, que con x número de scripts nos enfrentamos al mismo problema que con Ansible: pérdida de versatilidad frente a una situación no prevista.

2.3.3 Android/ios/aplicaciones de escritorio

Existen innumerables aplicaciones[9] para todos los ssoo pertenecientes a dispositivos móviles, para tanto controlar como configurar los drones. Habitualmente se hacen ad-hoc para una marca o un modelo. Si bien es cierto que no consumen casi nada de recursos y que pueden ser utilizadas por un usuario no experto, las aplicaciones generales o no son compatibles con el UAV o no poseen las funcionalidades necesarias. Por lo que quedan descartadas.

2.3.4 Monit/supervisor/daemontools

Los tres softwares [10] están focalizados en la supervisión de procesos con bastante versatilidad. Los tres son excelentes softwares que no consumen casi nada de recursos y, quitando a monit, no son más difíciles de usar que los comandos por terminal. Sn embargo, nos encontramos con el mismo problema que en todos los demás: no reúne todas las características que se nos plantean.

[7] Véase: Ansible (2019).
[8] Véase: Lehtinen, S. (2006)
[9] Véase 9.1 recursos electrónicos: Freeflight (2019); Pix4Dcapture (2019); Mission Planner (2019); Hover (2019).
[10] Véase: Monit (2019); Supervisor (2019); Bermstein, F.J. (2019)

A pesar de esto podríamos combinar alguno de estos tres con Ansible para cubrir todas las funcionalidades del problema planteado. Excepto que seguiría sin poder utilizarse por un usuario no experto.

2.3.5 Conclusión

Se nos hace evidente que no existe, en el mercado actual, ninguna aplicación, ni combinación de ellas, que satisfagan todas las características de este proyecto. Por lo tanto, se nos presenta la tarea de realizar un sistema ad-hoc desde cero para el UAV. Aunque, existe la posibilidad de que podamos integrar programas de terceros para completar algunas de las funcionalidades. Se trata en profundidad este tema en los siguientes apartados.

3 REQUISITOS

En los requisitos se recogen todas las funcionalidades y limitaciones que compondrán el software. Como este es un proyecto planificado, diseñado y desarrollado individualmente no se requiere de un seguimiento exhaustivo de los requisitos. Todos los requisitos estarán implementados y validados al final del proyecto, así que se da por hecho que se encuentran en estado "validado".

El formato en el que se detalla un requisito está inspirado en lo propuesto en el estándar IEEE 830[11]; no siendo este el mismo. Se eliminan campos innecesarios por las características únicas de este proyecto.

Un ejemplo de requisito sería.

ID: RF-xx	**Nombre descriptivo:** xxx	**Versión:** 04/03/2019
Descripción	Esto es una descripción.	
Comentarios	Esto es un comentario.	

Donde:

- **ID**: RF – requisito funcional; RS – requisito de sistema; RU – requisito de usuario.
- **Nombre descriptivo**: resumen en una o dos palabras del requisito.
- **Versión**: última modificación.
- **Descripción**: detalle de la funcionalidad o limitación.
- **Comentario**: información adicional o aclaración sobre el requisito o su descripción.

Los requisitos vienen impuestos por el INTA y por personal de la UC3M con 10 años de experiencia en la utilización del prototipo UAV con routers IP. Ambas fuentes de información contribuyen a la especificación de requisitos siendo estas dos indistinguibles. Como consecuencia se dice que los requisitos proceden tanto del INTA como del personal de la UC3M.

3.1 REQUISITOS FUNCIONALES

ID: RF-001	**Nombre descriptivo:** Conexión a IP/puerto	**Versión:** 04/03/2019
Descripción	El software deberá ser capaz de conectarse a la máquina UAV mediante una IP y un puerto.	
Comentarios	La IP y el puerto no estarán predeterminados.	

ID: RF-002	**Nombre descriptivo:** Cambiar IP/puerto	**Versión:** 04/03/2019
Descripción	El software deberá ser capaz de cambiar la IP y el puerto por el cual se conecta a la máquina UAV.	
Comentarios	---	

ID: RF-003	**Nombre descriptivo:** Mostrar IP	**Versión:** 04/03/2019
Descripción	El software deberá ser capaz de mostrar las IP y las interfaces que están presentes en la máquina UAV.	

[11] Véase: IEEE (2008)

Comentarios	---

ID: RF-004	**Nombre descriptivo:** Mostrar enrutamiento	**Versión:** 04/03/2019
Descripción	El software deberá ser capaz de mostrar información sobre la tabla de enrutamiento de la máquina UAV.	
Comentarios	La información que deberá mostrar es: • Destino • Pasarela • Máscara • Indicador • Métrica • Referencia • Uso • Interfaz	

ID: RF-005	**Nombre descriptivo:** Añadir enrutamiento	**Versión:** 04/03/2019
Descripción	El software deberá ser capaz de añadir una nueva línea de enrutamiento a la tabla de enrutamiento de la máquina UAV.	
Comentarios	Una línea de enrutamiento consiste en: • Destino • Pasarela • Máscara • Interfaz	

ID: RF-006	**Nombre descriptivo:** Borrar enrutamiento	**Versión:** 04/03/2019
Descripción	El software deberá ser capaz de borrar una línea de enrutamiento de la tabla de enrutamiento de la máquina UAV	
Comentarios	---	

ID: RF-007	**Nombre descriptivo:** Mostrar proceso	**Versión:** 04/03/2019
Descripción	El software deberá ser capaz de mostrar información relevante sobre un proceso de la máquina UAV.	
Comentarios	El proceso lo introduce el usuario. Se considera información relevante: • Estado • Número de hijos • UID • PID • PPID • Tiempo activo	

ID: RF-008	**Nombre descriptivo:** Lanzar proceso	**Versión:** 04/03/2019
Descripción	El software deberá ser capaz de lanzar o relanzar un proceso de la máquina UAV. También deberá dar la posibilidad de lanzar/relanzar con argumentos	

Comentarios	El proceso lo introduce el usuario. Lanzar se considera que es ejecutar un proceso que actualmente está muerto. Relanzar se considera que es ejecutar un proceso que actualmente no está muerto.

ID:	RF-009	**Nombre descriptivo:**	Matar proceso	**Versión:**	04/03/2019
Descripción	El software deberá ser capaz de matar un proceso de la máquina UAV.				
Comentarios	El proceso lo introduce el usuario. No es necesario recoger la salida del proceso matado. Se considera como cierre forzado y, por lo tanto, no se contempla la pérdida de información.				

ID:	RF-010	**Nombre descriptivo:**	Crear watcher	**Versión:**	04/03/2019
Descripción	El software deberá ser capaz de crear un watcher sobre un proceso de la máquina UAV.				
Comentarios	El proceso lo introduce el usuario. El watcher ya fue definido en el *1.6 Glosario de términos*.				

ID:	RF-011	**Nombre descriptivo:**	Eliminar watcher	**Versión:**	04/03/2019
Descripción	El software deberá ser capaz de eliminar un watcher sobre un proceso de la máquina UAV				
Comentarios	El proceso lo introduce el usuario. El watcher ya fue definido en el *1.6 Glosario de términos*.				

ID:	RF-012	**Nombre descriptivo:**	Mostrar watcher	**Versión:**	04/03/2019
Descripción	El software deberá ser capaz de mostrar la lista de watchers creados sobre procesos de la máquina UAV.				
Comentarios	El watcher ya fue definido en el *1.6 Glosario de términos*.				

3.2 REQUISITOS DE SISTEMA

ID:	RS-001	**Nombre descriptivo:**	SO	**Versión:**	04/03/2019
Descripción	El software deberá ser compatible con Ubuntu versión 14 en adelante, pues es el SO que tienen instalado las máquinas.				
Comentarios	---				

ID:	RS-002	**Nombre descriptivo:**	Disponibilidad máquinas	**Versión:**	04/03/2019
Descripción	Existe el Router de tierra y la máquina UAV. Solo se dispondrá de ambas.				
Comentarios	---				

ID:	RS-003	**Nombre descriptivo:**	Inaccesibilidad	**Versión:**	04/03/2019
Descripción	La máquina UAV no es accesible físicamente en una ejecución habitual. Es por esto que, se contactará con ella mediante ssh o UDP.				

Comentarios	Se utiliza el protocolo UDP debido a que la máquina UAV, por motivos de seguridad, rechaza algunos mensajes TCP[12].

ID:	RS-004	**Nombre descriptivo:**	Consumo recursos	**Versión:**	04/03/2019
Descripción	El software deberá consumir los mínimos recursos, tanto de procesamiento como de memoria. No se establece un máximo numérico, si no que el software no deberá saturar la máquina UAV impidiendo que otros procesos se ejecuten.				
Comentarios	---				

ID:	RS-005	**Nombre descriptivo:**	Escalabilidad (opcional)	**Versión:**	04/03/2019
Descripción	**Opcionalmente**, el software deberá ser escalable. Esto quiere decir que pueda funcionar para cualquier máquina con Ubuntu independientemente de qué marca sea el dron huésped.				
Comentarios	---				

3.3 REQUISITOS DE USUARIO

ID:	RU-001	**Nombre descriptivo:**	Interfaz sencilla	**Versión:**	04/03/2019
Descripción	El software deberá mostrarse a través de una interfaz intuitiva y usable para un usuario no especializado en campos de informática o telemática.				
Comentarios	El cliente no es un experto en el tema sobre el que se desarrolla el software.				

ID:	RU-002	**Nombre descriptivo:**	Instalación sencilla	**Versión:**	04/03/2019
Descripción	El software deberá poder instalarse con pocos pasos y sin instrucciones complicadas. Esto quiere decir que un usuario no especializado en informática sea capaz de instalarla.				
Comentarios	---				

[12] Véase: Cerf, Cinton. Dalal, Yogen. Sunshine, Carl. (1974)

4 DISEÑO

En diseño se distinguen dos niveles: interno e interfaz gráfica. El **diseño interno** engloba todo aquello que es transparente al usuario y que contiene la mayoría de funcionalidades que ya se especificaron en el apartado de requisitos. El **diseño de interfaz gráfica** engloba todo aquello que es 'estético' y que sirve de intermediario entre el usuario y el programa.

Se decide dividir en dos niveles el diseño ya que hasta no haber terminado la implementación del diseño a nivel interno no se puede comenzar el diseño de la interfaz gráfica. Esto es debido a que cambios en el diseño a nivel interno pueden acarrear cambios en la interfaz y, por lo tanto, rehacer la interfaz por cada cambio a nivel interno no es eficiente.

4.1 ALTERNATIVAS DE DISEÑO

A continuación, se presentan las alternativas de diseño más relevantes. Todas contemplan los requisitos funcionales. En este apartado se discuten los requisitos de sistema y de usuario que son más abiertos a interpretación. Al final se decide qué tipo de diseño encaja mejor con el problema planteado.

4.1.1 Alternativa 1: app – servidor

Se propone un sistema basado en una aplicación móvil para Android que se conecta a un servidor instalado en el propio UAV. Cuenta con una interfaz gráfica por la parte de la app y se conecta al servidor mediante su dirección IP. Se puede conectar al servidor desde cualquier parte dónde el dispositivo Android tenga acceso a internet, es por esto que, se deben implantar medidas adicionales de seguridad como puede ser un usuario y contraseña o medidas biométricas aprovechando el hardware de algunos dispositivos actuales. La parte del sistema de Android debe programarse en Android (Java), y la parte del servidor puede programarse en cualquier idioma. Dado que el UAV alberga un SO Linux, se propone programar el servidor en C++ por la versatilidad para optimizar los recursos de memoria y los recursos de procesamiento.

4.1.2 Alternativa 2: aplicación web – servidor

Se propone un sistema basado en una aplicación web que se conecta a un servidor instalado en el propio UAV. La aplicación web cuenta con una interfaz gráfica y se conecta al servidor mediante su dirección IP. Se puede conectar al servidor desde cualquier parte dónde el navegador tenga acceso a internet, es por esto que, se deben implantar medidas adicionales de seguridad como puede ser un usuario y contraseña. La aplicación web se alberga en un dominio y es compatible con los navegadores principales: Mozilla Firefox, Google Chrome, Safari y Microsoft Edge. La parte de la aplicación web se programará en HTML5, CSS3. La parte del servidor se pude programar en cualquier idioma. Dado que el UAV alberga un SO Linux, se propone programar el servidor en C++ por la versatilidad para optimizar los recursos de memoria y los recursos de procesamiento.

4.1.3 Alternativa 3: programa adaptativo

Se propone un sistema que pueda transformarse en cliente o en servidor dependiendo de la situación. Por lo tanto, ha de instalarse en dos dispositivos; en este caso el Router de tierra y el dron UAV. El cliente se conecta al servidor mediante su IP, pero tienen que estar dentro de la misma red. Cliente y servidor contarían con una interfaz gráfica, pues el servidor no está limitado a estar solo en el dron UAV. Se presupone que solo las personas autorizadas tienen acceso a la

red, por lo que no se implantan medidas de seguridad adicionales. El programa adaptativo puede programarse en cualquier lenguaje. Dado que el UAV y el Router de tierra albergan un SO Linux, se propone el lenguaje C++ por la versatilidad para optimizar los recursos de memoria y los recursos de procesamiento.

4.1.4 Alternativa 4: aplicación escritorio – servidor

Se propone un sistema similar a la alternativa 3, pero separando el cliente del servidor. Por lo tanto, el cliente existe en el Router de tierra y el servidor en el UAV. El cliente tiene una interfaz gráfica y se conecta al servidor mediante su IP, pero tienen que estar dentro de la misma red. Se presupone que solo las personas autorizadas tienen acceso a la red, por lo que no se implantan medidas de seguridad adicionales. El cliente y el servidor pueden programarse en cualquier lenguaje. Dado que el UAV y el Router de tierra albergan un SO Linux, se propone el lenguaje C++ por la versatilidad para optimizar los recursos de memoria y los recursos de procesamiento.

4.1.5 Justificación de diseño. Alternativa 4

Tabla 3 - Comparativa de diseños

Alternativa:	1	2	3	4
Compatible con Ubuntu	✗	✓	✓	✓
Fácilmente escalable	✗	✓	✗	✓
Interfaz versátil	✓	✓	✓	✓
Optimizado de recursos.	✗	✗	✓	✓
Instalación y ejecución sencilla	✓	✓	✓	✓
Rapidez	✗	✗	✓	✓

Todas las alternativas son válidas. Además, las alternativas 1 y 2 son compatibles entre sí y podrían servir de mejora a la alternativa 4. Sin embargo, teniendo en cuenta los factores cruciales de: a) solo existen el router de tierra y la máquina router UAV; b) debe ser obligatoriamente compatible con Ubuntu; c) debe ser escalable. La alternativa 2, 3 y 4 son las únicas posibles. Rechazamos la alternativa 1 porque la parte de la app debe ser compatible con Ubuntu, por lo que no satisface b). Rechazamos la alternativa 3 ya que se considera un diseño complejo y la alternativa 4 es similar y más escalable que la alternativa 3. Por lo tanto, quedan la alternativa 2 y 4. **Se selecciona la alternativa 4** pues se posee un control sobre los recursos[13] que se consumen y por la posible optimización de tiempos tanto para arranque como para tiempos de respuesta.

En la tabla se muestran, resumidos, la comparación según los requisitos más decisivos a la hora de elegir una alternativa u otra.

4.2 HERRAMIENTAS

Para el desarrollo e implementación de ambos niveles de diseño se van a utilizar las mismas herramientas y el mismo entorno. Se tiene en cuenta este factor para buscar el software más adecuado que permita tanto el desarrollo interno como el desarrollo de interfaz.

Lo primero que se decide es utilizar Java frente a C o C++ y otros lenguajes. Ambos, Java y C/C++ son lenguajes válidos para el proyecto y el resultado, de uno u otro, no debería variar significativamente. Los demás lenguajes complican el diseño o no son adecuados para la realización de interfaces. Se decide utilizar **Java** dada la experiencia y la rutina de programar en

[13] Recursos entendidos como: memoria y procesamiento.

este lenguaje. En concreto **openJDK 8** que es un software sencillo de instalar en plataformas Linux/Ubuntu y, además, gratuito.

Como entorno de desarrollo entregado (IDE) hay innumerables opciones: eclipse, NetBeans, IntelliJ IDEA y Oracle JDeveloper entre otros. Se decide utilizar **NetBeans 8.0** por:

- Ser una IDE enfocada a Java.
- Contar con un entorno para el desarrollo de interfaces.
- La posibilidad de extender la herramienta con 'módulos'.
- Compatible con GitHub.
- Compatible con Linux/Ubuntu.
- Gratuito y sin restricciones de uso.

Se ha descartado eclipse por tener un entorno para el desarrollo de interfaces de base no adecuado para el proyecto (no posee tanta versatilidad como el de NetBeans).

Se ha descartado IntelliJ IDEA por tener una versión de pago donde se libera la funcionalidad de un entorno para el desarrollo de interfaces.

Se ha descartado Oracle JDeveloper por tener un entorno para el desarrollo de interfaces de base no adecuado para el proyecto (no posee tanta versatilidad como el de NetBeans).

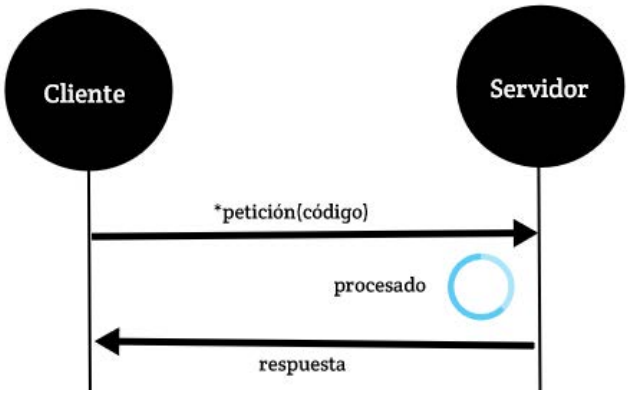
Como SO se decide utilizar Ubuntu 16.04, pues es el SO que se utilizará en las máquinas reales. Además, facilitará el desarrollo y la implementación que se realice con NetBeans ya que se podrá probar al instante el código en una máquina con las mismas características que la real.

4.3 DISEÑO: NIVEL INTERNO

El problema presenta dos máquinas y una de ella no es accesible físicamente. Es evidente que se requiere de una **arquitectura cliente-servidor** para este caso en concreto (alternativa 4). Además, si tomamos en cuenta el requisito: RS-003 Inaccesibilidad, nos damos cuenta que en cualquier caso siempre va a haber una máquina conectada a un Dron que no es accesible físicamente. La arquitectura cliente-servidor es perfecta: siendo el cliente la máquina conectada y el servidor la máquina del Dron (en este problema el UAV).

Siguiendo esta arquitectura, se diseñan un cliente y un servidor. El funcionamiento básico es el siguiente: el cliente realiza una petición al servidor; el servidor procesa la petición (realiza operaciones o lecturas de información) y devuelve un mensaje con lo solicitado en la petición; el cliente recoge esa información.

Ilustración 4 - Arquitectura cliente-servidor



Como vemos, la carga computacional reside en el servidor y el cliente se encarga de representar la información de forma gráfica (lo veremos en el siguiente apartado).

Se persigue esa escalabilidad a lo largo de todo el diseño. Es por esto que el cliente cuenta con una arquitectura modular para separar claramente las funcionalidades (y quizás en un futuro añadir nuevas). Por lo tanto, vemos dos aspectos de la escalabilidad que se trabajan: escalabilidad a otros sistemas y escalabilidad a otras funcionalidades. Por el contrario, el servidor no contará con una arquitectura modular. Debido a que muchas de las peticiones se procesan de la misma manera, las funcionalidades no están tan bien diferenciadas como en el cliente. También hay que tener en cuenta que el servidor ha de ser

robusto frente a errores, que, siendo el lenguaje de programación Java, esto se traduce a que la ejecución no termine nunca por una excepción.

4.3.1 Mensajes UDP

Para que el envío de mensajes se produzca, hay que crear un socket en el cliente y otro en el servidor. El socket requiere de una IP y un puerto. La IP del cliente se puede configurar, el puerto del cliente es elegido aleatoriamente entre los que no se utilizan. La IP del servidor y el puerto del servidor se configuran; aunque, existe el puerto predeterminado 4040, que, consultando la IANA[14] está asignado a una web que no se prevé utilizar en ninguna de las máquinas.

La conexión entre cliente y servidor solo se puede realizar por protocolo UDP[15] o por ssh (RS-003 Inaccesibilidad). Sin duda alguna es mucho más sencillo el desarrollo por mensajes UDP que por ssh; se consigue el mismo resultado o mejor, ya que se espera que UDP tenga una diferencia significativa en el tiempo de respuesta. Por lo tanto, las peticiones del cliente se enviarán como datagramas y las respuestas también. Se ha establecido un tamaño máximo de los datagramas que es 64KB. Es suficiente espacio para los mensajes que queremos enviar.

A las peticiones se les asigna un código arbitrario que reconoce el servidor y se concatenan con los posibles argumentos. Por lo tanto, el mensaje de petición consta de una cabecera de tamaño fijo y los posibles argumentos de tamaño variable.

Tabla 4 - Mensaje: petición

<petición>	<argumentos>?
8 bits	De 0 a 51192 bits

Siendo el código un char y los argumentos un String. A continuación, hay una tabla que resume todos los códigos asignados a peticiones.

Tabla 5 - Código de peticiones

Petición	Código	Descripción	Argumentos
Apagar servidor	0	El servidor deja de ejecutarse.	--
Consulta de IP	1	Consulta las IP disponibles (no incluye las de localhost ni las de broadcast)	--
Consulta de tabla de enrutamiento	2	Consulta la tabla de enrutamiento.	--
Cambiar IP servidor	3	El servidor cambia de IP y el cliente se reconecta a esa nueva IP	<IP>
Borrar línea de tabla de enrutamiento	4	Borra una línea de la tabla de enrutamiento	<línea>
Añadir línea de tabla de enrutamiento	5	Añade una línea a la tabla de enrutamiento	<línea>
Consultar proceso	6	Consulta la información de un proceso. La información está descrita en RF-007 Mostrar proceso	<proceso>

[14] Véase: IANA (2019)
[15] Véase: Postel, J. (1980)

Matar proceso	7	Mata a un proceso y todos sus hijos (killall)	<proceso>
Relanzar proceso	8	Lanza un proceso independientemente de si está corriendo o está muerto. Con opción a argumentos	<proceso> <argumentos>?
Crear watcher	9	Añade un watcher a la lista de watchers. Se crea un timer de 5 segundos para el watcher	<watcher>
Eliminar watcher	A	Elimina un watcher de la lista de watchers. Elimina el timer	<watcher>
Consultar si es un watcher	B	Consulta si hay un watcher vigilando el proceso	<proceso>
Consultar si existen permisos	C	Consulta si existen privilegios de root	--
Consultar timeout	D	Consulta el timeout	--
Cambiar timeout	E	Cambia el timeout	<timeout>
Consultar watchers	F	Consulta todos los watchers vigilando procesos y sus argumentos de relanzado	--
Consultar path de un proceso	G	Consulta el path de un proceso (es una forma de consultar si un proceso existe o no)	<proceso>
Consultar un proceso (PID y STAT)	H	Consulta solo la información relativa al PID y al STAT de un proceso y sus hijos	<proceso>

A las respuestas se les asigna un código de error [16] arbitrario que reconoce el cliente y se concatena con la información adicional. Por lo tanto, el mensaje de respuesta consta de una cabecera de tamaño fijo y la posible información de tamaño variable.

Tabla 6 - Mensaje: respuesta

<código de error>		<información>
error>	<petición>	
8 bits	8 bits	
16 bits		De 8 a 51184 bits

El código de error se detallará en el apartado *4.3.6 Tratamiento de errores*. Adelantamos que el código de error de que no ha habido ningún error es "00". Por lo tanto, las respuestas satisfactorias serán del estilo: 00<información>.

Tabla 7 - Respuesta del servidor

Petición	Código	Respuesta
Apagar servidor	0	HECHO
Consulta de IP	1	<IP>
Consulta de tabla de enrutamiento	2	<tabla de enrutamiento>
Cambiar IP servidor	3	HECHO
Borrar línea de tabla de enrutamiento	4	HECHO
Añadir línea de tabla de enrutamiento	5	HECHO

[16] Que no haya habido un error se contempla dentro del código de errores

Consultar proceso	6	<info. proceso>
Matar proceso	7	HECHO
Relanzar proceso	8	HECHO
Crear watcher	9	HECHO
Eliminar watcher	A	HECHO
Consultar si es un watcher	B	true/false
Consultar si existen permisos	C	true/false
Consultar timeout	D	<timeout>
Cambiar timeout	E	HECHO
Consultar watchers	F	<watchers>
Consultar path de un proceso	G	<path>
Consultar un proceso (PID y STAT)	H	<PID><STAT>

4.3.2 Cliente (CC_tierra.jar)

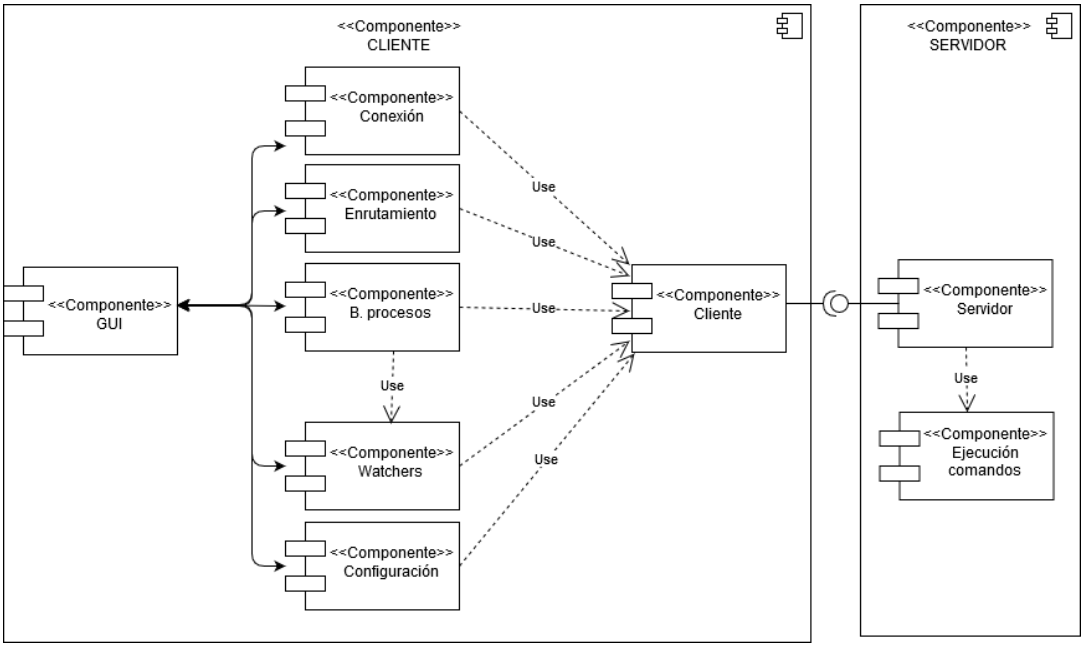
El cliente es el único que tiene apartado de interfaz gráfica. A nivel interno cuenta con diferentes módulos que recogen funcionalidades bajo un mismo título. Estas son:

- **Cliente**: se encarga de proveer los códigos para peticiones, de enviar peticiones, de recibir respuestas y de leer los códigos de error en las respuestas. Alberga el socket.
- **Conexión**: se encarga de conectar cliente y servidor en una IP y puerto en concreto. Además, desconecta cliente del servidor y apaga el servidor.
- **Enrutamiento**: se encarga de realizar operaciones (añadir o borrar) en la tabla de enrutamiento del servidor y de consultar información de la tabla de enrutamiento.
- **Buscador de procesos**: se encarga de buscar si un proceso existe y consultar información al respecto. Puede matar y relanzar, con o sin argumentos, los procesos del servidor. Además muestra información relativa a los watchers[17].
- **Watchers**: se encarga de crear, eliminar y consultar información sobre los watchers del servidor.
- **Configuración**: se encarga de modificar la IP del servidor, el puerto del servidor, el timeout del servidor, la IP del cliente, el timeout del cliente.

Como podemos comprobar en el diagrama de componentes, el componente *Cliente* interacciona con el servidor para consultar información o solicitar operaciones y todos los demás módulos utilizan esta vía para realizar sus funciones. Asimismo, adelantamos que la GUI (o interfaz gráfica) se nutre de todos los módulos, menos el Cliente, para representar la información.

[17] Si bien es cierto que *Watchers* y *Buscador de procesos* tienen un enlace entre ellos, ya que el *Buscador de procesos* utiliza parte de las funcionalidades de *Watchers*. Se consideran independientes pues el *Buscador de procesos* utiliza al módulo *Watchers* para realizar funciones propiamente de *Watchers*, como es la de crear o eliminar un watcher. Se ha introducido este enlace porque el watcher es asignado a un proceso y, por lo tanto, es información sobre un proceso que debe de ser mostrada en el *Buscador de procesos*.

Ilustración 5 - Diagrama de componentes



4.3.3 Servidor (CC_aire.jar)

El servidor 'escucha' peticiones del cliente, procesa esas peticiones (operación o consulta de información) y devuelve una respuesta. El cuerpo de esta respuesta ya se ha explicado en el apartado *4.3.1 Mensajes UDP*. Además, en segundo plano, cada 5 segundos comprueba los wacthers que se hayan creado, se explicarán al detalle en el siguiente apartado. El servidor se ocupa de almacenar estos watchers para la siguiente ejecución. Dada su función con los watchers, el servidor está pensado para funcionar continuamente para realizar la comprobación de wacthers. Loo que quiere decir que una vez que terminemos de configurar o consultar información desde el cliente, este se desconecta y no apaga el servidor.

Las peticiones se pueden englobar en tres tipos:

- Consultar información.
- Borrar:
- Crear/añadir.

Salvo en el caso de los watchers, que se mantienen como datos internos, las peticiones se ejecutan con comandos a través bash.

4.3.4 Watchers

La finalidad de un watcher es relanzar un proceso si se detecta que está muerto. Se presupone que se crea un watcher sobre un proceso para 'vigilar' (de ahí su nombre) que siempre esté corriendo. Es por esto que los watchers quedan en la parte del servidor, para vigilar sus procesos; si enviásemos continuamente peticiones desde el cliente de consulta de estado de un proceso y petición de relanzar un proceso afectaría negativamente al tiempo de respuesta y saturaría innecesariamente el canal de comunicación.

Un watcher se compone de: el nombre de un proceso y los argumentos para relanzarlo. Se puede crear un watcher sobre un proceso que no exista, pues en plena ejecución es posible que se desinstalen programas y se olvide quitar el watcher.

El funcionamiento de un watcher es el siguiente:

- Si el proceso no existe o ya está corriendo: no hace nada.
- Si el proceso existe y está muerto: lo relanza con los argumentos que se especificaron.

4.3.5 Permisos

Las operaciones sobre la tabla de enrutamiento requieren privilegios root. Es por esto que se pide que se ejecute el servidor con los privilegios root, ya que, si se garantiza al servidor estos privilegios, sus operaciones heredarán los privilegios y se podrá modificar la tabla de enrutamiento.

4.3.6 Tratamiento de errores

Se decide tener una robustez frente a errores por la función crítica del sistema en tiempo real. La robustez se traduce en: tratamiento de errores y recuperación frente a errores. Estas medidas se realizan de forma distinta en el cliente y en el servidor.

4.3.6.1 Cliente

Hay dos tipos de errores que se tratan.

El primero es una desconexión con el servidor inesperada (usualmente se debe a un timeout) el tratamiento es desactivar los módulos de: Enrutamiento, Buscador de procesos, Watchers y Configuración. Con esto conseguimos que solo se pueda utilizar el módulo de conexión para volver a establecer una vía de comunicación. Es evidente que sin esta vía no se pueden hacer peticiones por lo que las otras funcionalidades quedan desactivadas.

El segundo tipo de error son los que devuelve el servidor. Ya se explicó en el apartado *4.3.1 Mensajes UDP* que en la respuesta del servidor hay una cabecera que con el código de error. A continuación, se detallan todos los posibles errores. La consecuencia es que la petición no se ha cumplido de forma satisfactoria, pero no se desactivan los módulos mencionados anteriormente.

Tabla 8 - Código de errores - motivo

Código de error	Petición relacionada	Motivo
00	--	Sin error.
01	**Consulta de IP**	No existen interfaces de red o no se han podido leer las interfaces de red.
02	**Consulta de tabla de enrutamiento**	No existe tabla de enrutamiento o no se ha podido leer la tabla de enrutamiento.
03	**Cambiar IP servidor**	No se ha podido cambiar la IP del servidor.
04	**Borrar línea de tabla de enrutamiento**	No se ha podido eliminar la línea de enrutamiento: no existe tal línea.
05	**Añadir línea de tabla de enrutamiento**	No se ha podido añadir la línea a la tabla de enrutamiento: línea no válida
06	**Consultar proceso**	No se ha podido obtener información del proceso: proceso inexistente.
07	**Matar proceso**	No se ha podido matar el proceso: no existe o no se tiene los privilegios necesarios.
08	**Relanzar proceso**	No se ha podido relanzar el proceso: no existe el proceso o argumentos inválidos.
09	**Crear watcher**	No se ha podido crear el watcher: memoria llena.

0A	**Eliminar watcher**	No se ha podido eliminar el watcher: watcher no encontrado.
0B	**Consultar si es un watcher**	No existe una lista de watchers.
1B		La lista de watchers está vacía. (previene futuras consultas)
0C	**Consultar si existen permisos**	No se ha podido consultar si existen privilegios root: respuesta inesperada.
0D	**Consultar timeout**	No se ha podido consultar el timeout: no existe socket.
0E	**Cambiar timeout**	No se ha podido cambiar el timeout: no existe socket.
0F	**Consultar watchers**	La lista de watchers está vacía o no existe (previene futuras consultas)
0G	**Consultar path de un proceso**	No se ha podido consultar el path.
0H	**Consultar un proceso (PID y STAT)**	No se ha podido consultar el PID y STAT.
99	**--**	Petición rechazada: el servidor no ha reconocido la petición

4.3.6.2 Servidor

Es importante que bajo ningún motivo deje de correr. En el servidor se tratan todos los errores que involucren ejecución de comandos o lectura de datos. En caso de que alguno devuelva una excepción se devolverá un error al cliente. El único caso en el que se ejecutan comandos sin haber sido una petición del cliente son en las comprobaciones de watchers cada 5 segundos. En caso de que un relanzado falle o tarde más de lo debido, se ignora y se intentará en el siguiente tick de 5 segundos.

4.3.7 Instalación

Tres pasos:

1. Instalar Java 8 o superior en ambas máquinas. Se recomienda utilizar openJDK8 que se puede instalar fácilmente ejecutando el siguiente comando en la terminal mientras se tiene una conexión a internet:

   ```
   sudo apt-get install openjdk-8-jre
   ```
2. Crear una carpeta en el *Router de tierra* y copiar el archivo CC_tierra.jar dentro.
3. Crear una carpeta en la *Máquina UAV* y copiar el archivo CC_aire.jar dentro.

4.3.8 Ejecución

Para el cliente:

```
Java –jar <path de CC_tierra.jar>
```

Para el servidor:

```
sudo Java –jar <path de CC_aire.jar> -i <IP> -p <puerto> -t <timeout>
```

*los flags de puerto y timeout no son obligatorios.

*Si se utiliza una sesión ssh para ejecutar el servidor hay que tener en cuenta que una vez cerrada la sesión ssh el servidor debe seguir corriendo. Por lo que se debe utilizar el comando `screen` (o similares) al final del comando de ejecución del servidor para mantenerlo corriendo una vez que la sesión ssh se cierre.

4.4 DISEÑO: INTERFAZ GRÁFICA

En este apartado se trata la **distribución** y los tipos de elementos que están en la interfaz gráfica. El aspecto estético, como el color, la forma y los bordes se tratarán en el apartado 5.3 Desarrollo: interfaz gráfica. El servidor es completamente transparente para el usuario, por lo tanto no tiene interfaz gráfica; el cliente es el que implementa una interfaz gráfica.

4.4.1 Aclaración de características

Las características de la interfaz gráfica se tienen que deducir a través del requisito RU-001 Interfaz sencilla, que nos detalla que ha de ser una interfaz **intuitiva** para un **usuario no especializado**. El contenido de la interfaz son todos los demás requisitos funcionales. Debido a lo anterior, lo que se busca en esta interfaz gráfica es: 1) Utilidad; 2) Efectividad; 3) Eficiencia; 4) Seguridad; 5) Usabilidad; 6) Interactividad.

4.4.1.1 Utilidad

Se refiere a qué puede hacer el sistema; las funciones son las que necesita el usuario. Hay dos funciones principales: a) mostrar información; b) realizar operaciones (crear/añadir y borrar) en este sistema.

4.4.1.2 Efectividad

Se refiere a identificar el botón, icono, campo de texto… con la función que realiza. También se puede entender de la forma inversa: querer realizar una función e identificar qué elemento gráfico la realiza.

4.4.1.3 Eficiencia

Se refiere a disponer de los elementos más utilizados de la forma más rápida. La interfaz representa a los usuarios lo necesario para llevar a cabo sus tareas de una forma rápida.

4.4.1.4 Seguridad

Se refiere a prevenir al usuario de realizar operaciones indeseables o condiciones peligrosas.

4.4.1.5 Usabilidad

Engloba varios conceptos[18]:

- Fácil de aprender para un usuario nuevo: reconocimiento antes que recuerdo.
- Fácil de usar para un usuario nuevo: pocos pasos para realizar una tarea.
- Robustez: apoyo al usuario para completar una tarea

Las cuatro primeras características (utilidad, efectividad, eficiencia y seguridad) influyen en que este sistema sea usable o no[19].

[18] Véase: ISO (2018)
[19] Véase: Beth Rosson, Mary. M.Carroll, John (2002).

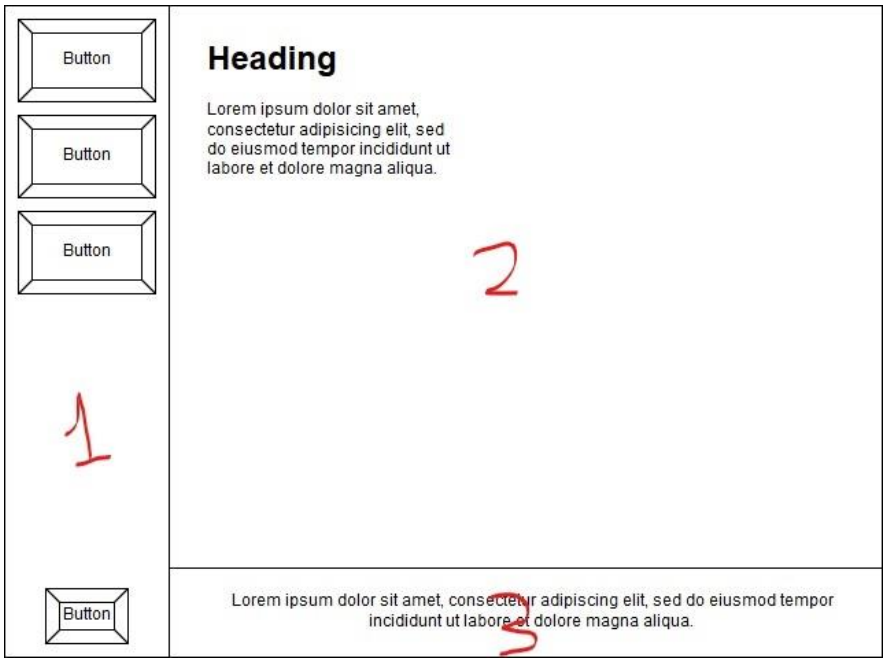
4.4.1.6 *Interactividad*[20]

Se refiere a que la comunicación se produce en ambos sentidos. Cada acción del usuario genera un cambio visual en la interfaz.

4.4.2 Wireframes

A continuación, se detallan las diversas vistas que posee la interfaz gráfica. Vamos a empezar identificando las diferentes zonas delimitadas en 3 rectángulos.

Ilustración 6 - Wireframe: vista general



1. Menú
2. Zona del módulo
3. Zona de información

Distinguimos entre dos tipos de frames: los estáticos y los dinámicos. Los estáticos no cambian de vista, pero sus elementos internos si pueden ser modificados. Los dinámicos sí cambian de vista y sus elementos internos pueden ser modificados. Frames estáticos son: 1) Menú 2) Zona de información. Frames dinámicos son: 3) Zona del módulo.

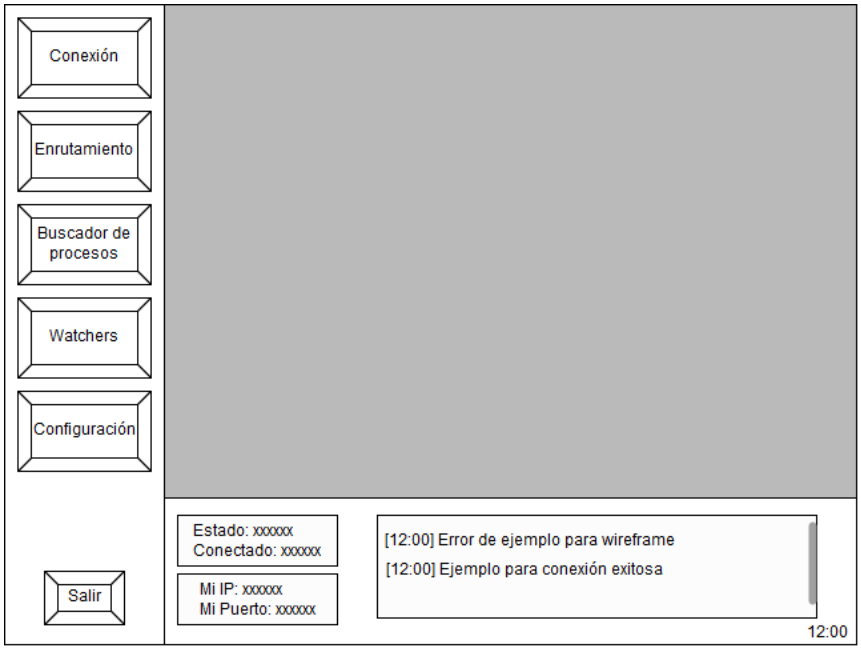
4.4.2.1 *Frames estáticos: Menú y Zona de información*

El menú nos facilita el acceso a cualquiera de las vistas de la Zona del módulo, permitiéndonos navegar con total libertad. Una lista es la mejor alternativa para pocas opciones a enseñar al mismo tiempo: no satura y se identifica con facilidad. Además, se aparta la opción de salir y se le da otro aspecto, para evitar posibles errores.

La zona de información no necesita ser interactiva. Muestra la información relevante a la conexión actual entre cliente y servidor; y muestra la información sobre posibles errores. La información relativa a la conexión se coloca más cercana al menú, lo que facilita su visibilidad. La información de error se coloca al extremo derecho, lo que dificulta su visibilidad, pues se consulta de forma ocasional.

[20] Véase: Preece, J., Rogers, Y., & Sharp, H. (2002).

Ilustración 7 - Wireframe: Menú y Zona de inf.



Utilidad

- Navegar
- Salir
- Conocer el estado de conexión

Efectividad

- Palabras en vez de iconos

Eficiencia

- Menú agrupado en lista
- Información de estado e información de error separada

Usabilidad

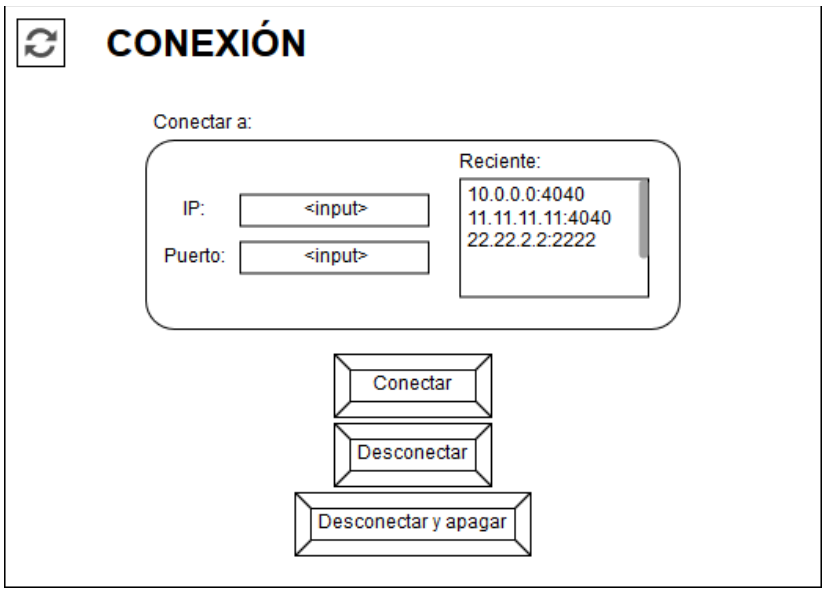
- Fácil de usar
- Fácil de aprender

4.4.2.2 *Frame dinámico: Zona del módulo*

La zona del módulo se divide en cinco vistas que se van alternando según el momento. Estas vistas son homónimas a los botones del menú. Habiendo, así, cinco: Conexión, Enrutamiento, Buscador de procesos, Watchers y Configuración. Todos ellos tienen en común la distribución de la parte superior. Se explica por primera vez en conexión y se aplica a las demás vistas sin volverlo a mencionar.

Conexión

Ilustración 8 - Wireframe: Conexión



Utilidad

- Des/conectar con el servidor
- Apagar servidor

Efectividad

- Palabras en vez de iconos

Eficiencia

- IP y puerto agrupado
- Acciones(botones) separados

Usabilidad

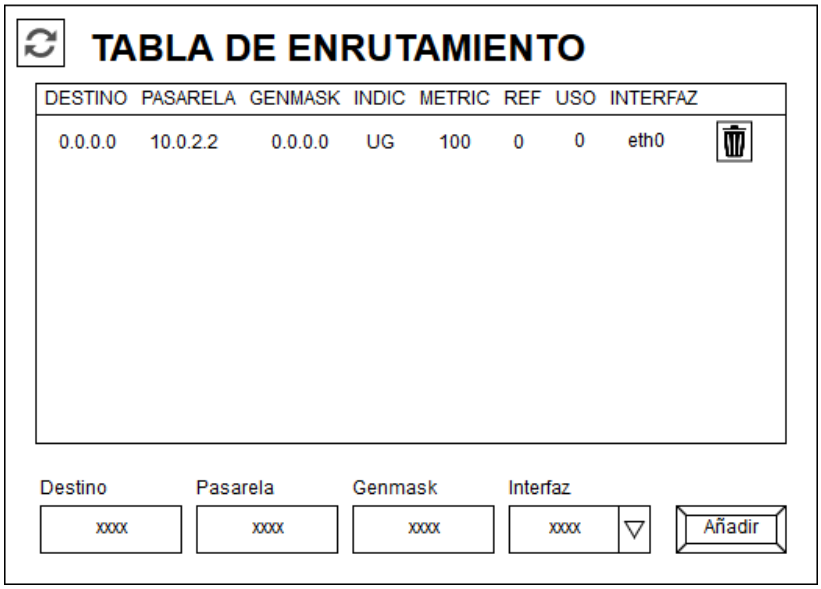
- Fácil de usar
- Fácil de aprender

En la parte superior observamos un icono y el título de la vista. El icono significa refrescar, lo que hace que recargue la vista a su modo predeterminado y descarte los cambios que se hayan producido.

En la parte central observamos un recuadro con el input de IP y puerto a introducir por el usuario y justo a su derecha (facilitando su visibilidad) las últimas IP y puertos utilizados. Siguiendo con la parte más baja de la vista, hay tres botones ordenados por frecuencia de utilización: arriba el más utilizado.

Enrutamiento

Ilustración 9 - Wireframe: Enrutamiento



Utilidad

- Mostrar tabla de enrutamiento
- Operaciones en tabla de enrutamiento

Efectividad

- Tabla ordenada con elementos separados
- Icono de *papelera* para eliminar

Eficiencia

- Tabla y operaciones separadas
- Ordenado los campos de añadir

Usabilidad

- Fácil de usar
- Fácil de aprender

Ocupando la mayor parte del espacio se muestra la tabla de enrutamiento. Se ajusta el ancho para que se pueda mostrar con claridad todos los campos. El icono de papelera se coloca a la altura de la línea indicando así sobre qué línea recae la operación de borrado. Lo mismo ocurre con la operación de añadir.

Buscador de procesos

Se ordenan los elementos de arriba abajo, siendo arriba el más visible y abajo el menos visible. Lo más utilizado será el buscador que se identifica por el campo de texto editable y el icono de una lupa; seguido, por relación en funcionalidad, están las búsquedas recientes. Cuando se busca un proceso, se rellena la información correspondiente en el recuadro central dejando todas las acciones posibles del usuario en la zona inferior del mismo recuadro. En el último recuadro observamos los elementos necesarios para relanzar un proceso.

Utilidad

- Mostrar información procesos
- Crear/borrar watchers
- Relanzar procesos

Efectividad

- Recuadros para cada utilidad
- Botones con texto
- Icono *lupa* para buscar

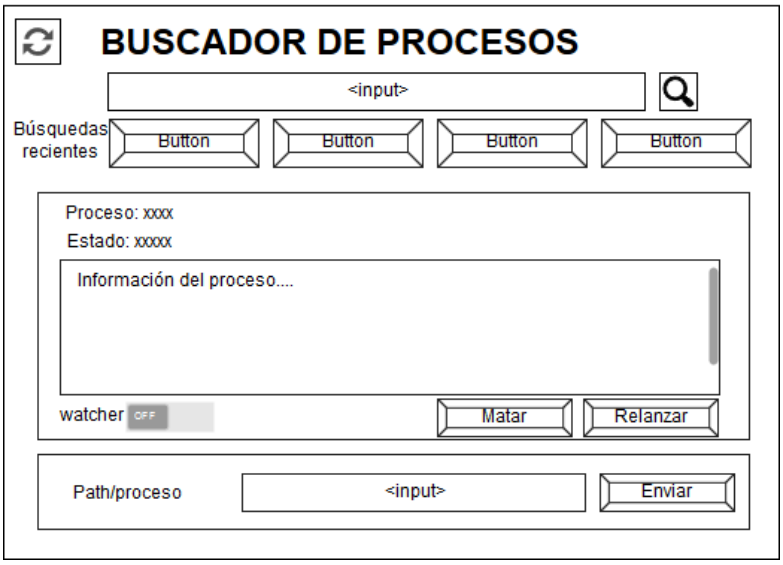
Eficiencia

- Tabla y operaciones separadas
- Ordenado los campos de añadir

Usabilidad

- Fácil de usar
- Fácil de aprender

Ilustración 10 - Wireframe: Buscador de procesos



Watchers

Ilustración 11 - Wireframe: Wathcers

Utilidad

- Mostrar watchers
- Crear/borrar watchers

Efectividad

- Tabla ordenada con elementos separados
- Icono *papelera* para eliminar

Eficiencia

- Tabla y operaciones separadas
- Ordenado los campos de añadir

Usabilidad

- Fácil de usar
- Fácil de aprender

Existen dos recuadros diferenciados. Una distribución similar a la vista de Enrutamiento: una tabla que muestra toda la información (con el icono de la papelera para la operación de eliminar) y una zona inferior para la operación de añadir.

Configuración

Utilidad

- Configuración IP, puerto y timeout servidor
- Configuración timeout servidor

Efectividad

- Nombres identificativos para cada elemento

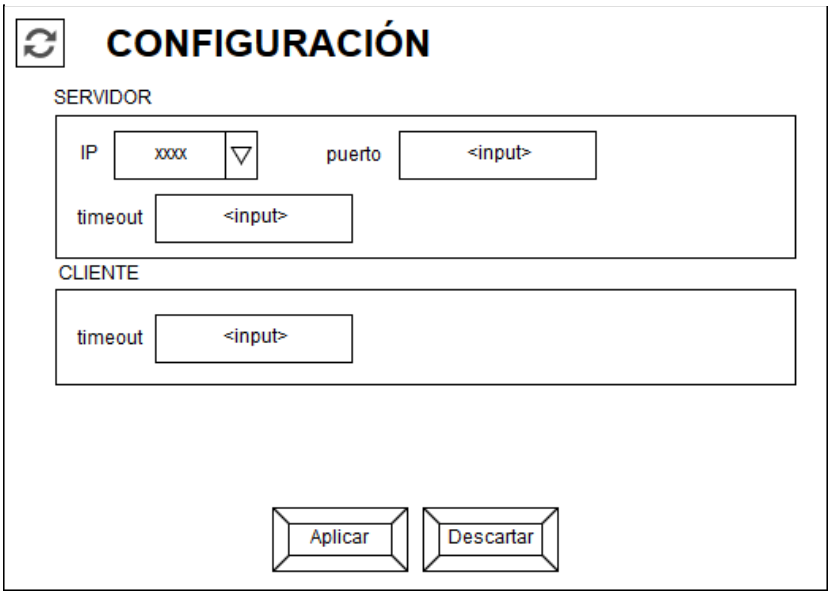
Eficiencia

- Ordenado por frecuencia de uso esperado
- Botones agrupados en el inferior

Usabilidad

- Fácil de usar
- Fácil de aprender

Ilustración 12 - Wireframe: Configuración



La última vista consiste en ajustas diversas variables del servidor y el cliente. Se separan en dos recuadros con título las variables que corresponden a cada uno y se dejan todas las acciones (en forma de botón) abajo para una vez configurado todo poder aplicar los cambios.

4.4.3 Estados

Hay 3+2 estados posibles del software; 3 son globales del software y 2 se utilizan para una vista en concreto; de ahí el 3+2. Además, son independientes entre sí por lo que siempre habrá una combinación de 2 estados en el software uno proveniente de los 3 y otro de los 2.

3 estados:

- **Conectado**: cliente y servidor poseen una vía de comunicación.
- **Desconectado**: cliente y servidor no poseen una vía de comunicación.
- **Procesando**: esperando respuesta del servidor o procesando una tarea interna de lectura.

+2 estados:

- **Con privilegios[21] root**: el servidor posee privilegios root.
- **Sin privilegios root**: el servidor no posee privilegios root.

[21] Ya destacamos la importancia de los privilegios root y cómo conseguirlos en el apartado *4.3.5 Permisos*

4.4.4 Restricciones

El flowchart de las vistas es bastante sencillo, desde cualquier vista puedes llegar hasta cualquier otra vista, por lo que lo omitiremos. Lo que nos interesa son las restricciones de acceso o no a una vista u elementos dentro de una vista según las restricciones. Las restricciones pueden estar causadas por el estado del software o por la respuesta a una petición; éstas son:

- **Habilitado**: sin restricciones.
- **Deshabilitado**: permanece visible pero no se puede interactuar, a pesar de ser un elemento interactivo (véase botón o campo de texto editable)
- **Bloqueado**: el software no genera ningún tipo de interactividad.

La restricción de 'bloqueado' solo aparece cuando el sistema se encuentra en estado 'procesando'. Veremos, a continuación, donde se aplica la restricción de deshabilitado. Por defecto todo está habilitado

Existe una restricción en las vistas: Cuando el sistema está desconectado, se cambia automáticamente a la vista de Conexión y se deshabilitan las demás opciones del menú, exceptuando conexión y salir.

Ahora enumeramos las restricciones de elementos gráficos agrupadas por vista:

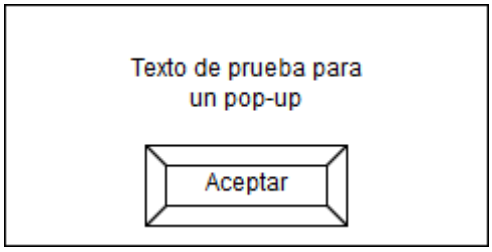
- Conexión
  - Si el sistema está desconectado, se deshabilitan los botones de *desconectar* y *desconectar y apagar*
  - Si el sistema está conectado, se deshabilita el botón de *conectar*
- Enrutamiento
  - Si el sistema no posee privilegios root, se deshabilitan los botones con el icono de *papelera* y el botón de añadir.
- Buscador de procesos
  - Si el proceso no existe, se deshabilita el switch de watcher y los botones de *matar* y *relanzar*
- Watchers – sin restricciones en elementos gráficos
- Configuración – sin restricciones en elementos gráficos

4.4.5 Pop-ups y Mensajes de errores

Informar de un error puede ayudar al usuario a identificarlo y resolverlo. Se utilizan las cajas de dialogo (en forma de pop-up) ya que muestran una información, pero se considera secundaria. Un pop-up hace que el sistema se bloquee excepto la zona del pop-up.

Un ejemplo sería:

Ilustración 13 - Pop-up



Los pop-up aparecen cuando: a) se genera un error tanto por una excepción interna como por una respuesta de error del servidor; b) se pulsa el botón de salir y el servidor no está apagado; c) desde la vista de *Buscador de procesos* se crea un watcher. En el caso b) se le

pedirá si quiere apagar el servidor antes de irse con las opciones de Sí, no y cancelar. En el caso c) el pop-up pedirá al usuario si quiere añadir argumentos.

4.5 MARCO REGULADOR LEGAL

Existe un aspecto de estándares que hay que respetar a la hora de programar; y existe un aspecto legal de la licencia sobre el producto final, es decir, el software generado.

Los estándares que se han de respetar son limitaciones, pautas y en algunos casos consejos aplicables al entorno de programación:

- Protocolo UDP: Estándar RFC 768.
- Estándar de programación ética: Code of Ethics and Professional Practice Javier Dolado (1999).

La licencia que se aplica al producto resultante de este proyecto es: **GNU General Public License**[22]. Lo que significa que es software libre, código abierto y cualquier usuario puede usar, modificar y compartir el código. Los autores sucesores deberán aplicar esta licencia protegida con copyleft y el software no podrá ser apropiable. Además, no es compatible con otras licencias, por lo que solo se aplicará esta. Sin embargo, software libre no es antónimo a comercializable.

La licencia que se aplica a la documentación del proyecto es: **Creative Commons BY-NC-ND**[23], otra licencia de la mano del copyleft. Lo que significa que: 1) requiere de la referencia del autor (BY); 2) No es comerciable (NC); 3) no se puede modificar la obra (ND).

4.6 ENTORNO SOCIO-ECONÓMICO

Al ser un software ad-hoc para la empresa del INTA, se espera un impacto mínimo en la sociedad. Por el contrario, se espera un efecto positivo en el INTA. Además, debido a la característica de escalabilidad, con un futuro diseño y ampliación, se podría convertir en un programa propio para los drones del INTA. Esto reforzaría la colaboración entre la UC3M y el INTA quizás requiriendo más proyectos o estándares en sus drones.

Con poca probabilidad, el software se podría escalar a otras marcas, modelos y convertirlo en un estándar para la configuración de drones. La manera en la que veo que esto se pueda cumplir es habilitando la posibilidad de plugíns y ofreciendo una api pública para que se forme un ecosistema de funcionalidades alcanzando todas las marcas y modelos. Persiguiendo siempre la mentalidad de software libre. Por supuesto, a este nivel, la financiación es un arma de doble filo ya que cualquiera puede modificar el software y programar una mejor versión. Y, además, los usuarios han de esforzarse para crear sus plugins sin ningún tipo de beneficio esperado. Por lo que, antes de dar este paso se tiene que realizar un estudio más concienzudo del destino del proyecto.

No se espera ningún efecto adicional.

[22] Véase: Free Software Foundation, Inc. (2007)
[23] Véase: Aliprandi, Simone (2011)

4.7 TRAZADO DE REQUISITOS

Módulo	RF-01	RF-02	RF-03	RF-04	RF-05	RF-06	RF-07	RF-08	RF-09	RF-10	RF-11	RF-12
Cliente	✓	✓			✓	✓		✓	✓	✓	✓	
Conexión	✓											
Enrutamiento			✓	✓	✓	✓						
Buscador de procesos							✓	✓	✓			
Watchers										✓	✓	✓
Configuración		✓										
Interfaz gráfica												
FINAL	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓

5 DESARROLLO

En este apartado se refleja cómo se desarrolla el código y la interfaz gráfica siguiendo el diseño. Se detallan, además, los problemas encontrados, junto a sus soluciones. Se divide en tres apartados: un primero con el entorno de trabajo con el cual se ha llevado a cabo el desarrollo; y otros dos profundizando en el nivel interno y en la interfaz gráfica respectivamente.

Todo el código fuente y las versiones se han subido a la plataforma GitHub con los enlaces[24] permanentes:

https://github.com/Romeorubiko/CCTierra

https://github.com/Romeorubiko/CCAire

5.1 ENTORNO DE TRABAJO

Para llevar a cabo el desarrollo del programa se habilita un entorno de trabajo adecuado. Aprovechando el laboratorio en las instalaciones de la universidad, se instala ahí el equipamiento. Consiste en:

- Mesa y casillero (para almacenar el equipamiento) en el laboratorio
- Ordenador de sobremesa
- Máquina Router UAV 1
- Máquina Router UAV 2

Se nos presenta el primer problema que es: adaptar el entorno de trabajo; tanto para poder realizar el desarrollo como para hacer eficiente el desarrollo.

5.1.1 Acomodación de entorno de trabajo

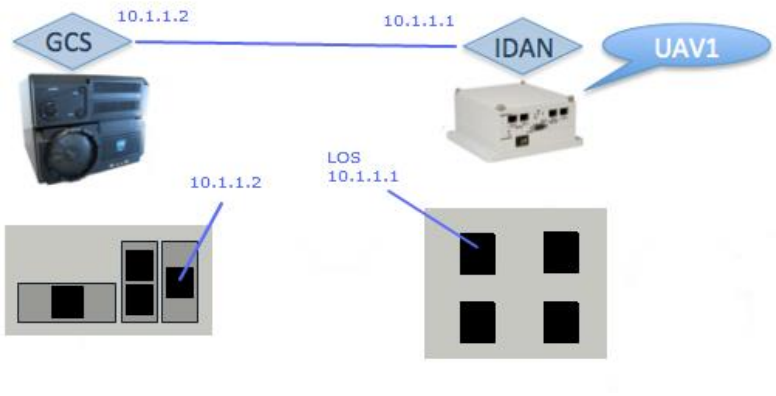
El ordenador de sobremesa no posee una gran potencia en comparación con los ordenadores de hoy en día. Sin embargo, esto no dificulta el trabajo pues NetBeans, que es la IDE que se utiliza para programar, no consume excesivos recursos y nunca se sobrecargará por ese motivo.

Tanto la *Máquina Router UAV 1* como la *Máquina Router UAV 2* no funcionaban. Seguramente era porque la última persona que trabajó con ellas las configuró para que funcionaran en otro entorno. Es por esto que, se formatean ambas máquinas y se instala el SO Ubuntu server 16.04.05 LTS de 64 bits, ya que un Ubuntu que no fuera de tipo servidor suponía mucha carga computacional para la *Máquina Router UAV 1 y 2*. Y un Ubuntu de 32 bits no posee la potencia necesaria.

Se termina utilizando **solo** la *Máquina Router UAV 1*; es indiferente utilizar una u otra. Se configura la conexión *ssh* en el puerto 2222 para que escuche mensajes desde la IP: 10.1.1.2 siguiendo el esquema de red planteado de forma arbitraria que dejó la última persona que trabajó con ella. Es posible utilizar esquemas diferentes, pero el resultado será el mismo. Se decide utilizar este en concreto:

[24] Al ser dos partes claramente diferenciables y que generan ejecutables distintos, se decide abrir dos repositorios para cada cliente y servidor respectivamente.

Ilustración 14 - Esquema de red



Como observamos en la imagen, al ordenador de sobremesa se llama GCS (*ground control system*) y a la *Máquina Router UAV 1*, se llama IDAN; a partir de ahora nos referiremos a ellos como GCS e IDAN.

GCS e IDAN en un principio se conectan a internet para descargar Java 1.8 y NetBeans 8.0, pero no requerirán de esa conexión para nada más. Por lo que la fase de desarrollo y la de pruebas se llevaron a cabo únicamente conectando los equipos mediante el *Switch* y dos cables ethernet utilizando las clavijas configuradas previamente en el esquema de red.

GCS alberga CC_tierra (cliente)

IDAN alberga CC_aire (servidor)

El equipamiento final queda así:

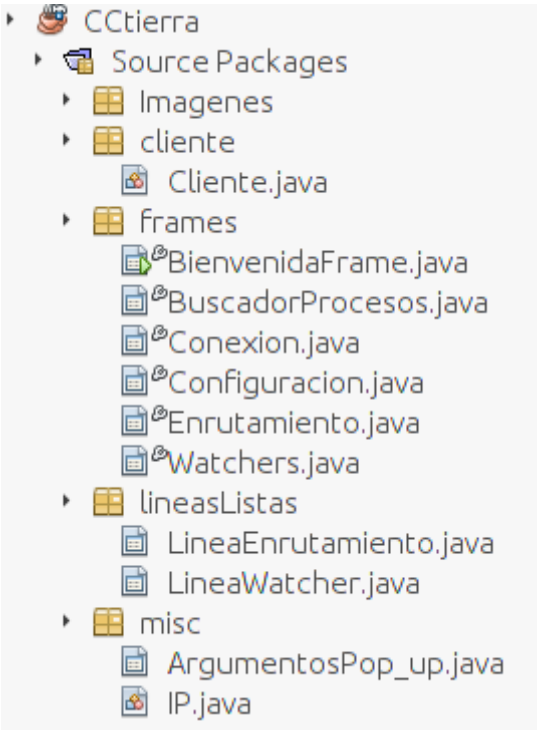
- Mesa y casillero (para almacenar el equipamiento) en el laboratorio
- Ordenador de sobremesa
  - Ubuntu 16.04
  - NetBeans 8.0
  - Java 1.8: openjdk8
- Máquina Router UAV 1
  - Ubuntu server 16.04.05 LTS
  - Java 1.8: openjdk8
- Máquina Router UAV 2
  - Ubuntu server 16.04.05 LTS
  - Java 1.8: openjdk8
- Switch
  - Cables ethernet

5.2 DESARROLLO: NIVEL INTERNO

A continuación, se exponen las características y las justificaciones del código desarrollado siguiendo las pautas de diseño previamente definidas. Se han tomado decisiones en base a: a) el diseño; b) el objetivo del software.

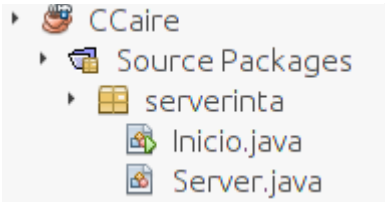
Persiguiendo la estructura modular (en la parte del cliente), para cada módulo se define una clase en Java de nombre homónimo.

Ilustración 15 - jerarquía de clases: cliente



Por lo tanto, cuenta con 6 clases principales (`Cliente.Java`, `BuscadorProcesos.Java`, `Conexión.Java`, `Configuracion.Java`, `Enrutamiento.Java`, `Watchers.Java`) y las demás sirven de apoyo o son derivados de los módulos. Es importante destacar que, durante el desarrollo del nivel interno, se desarrolla de manera simultánea una interfaz gráfica muy básica que se compone de los mismos elementos gráficos que se especificaron en el diseño, pero no necesariamente de la misma distribución y/o apariencia. Agiliza el proceso de programación dado que los cambios se pueden probar en un entorno simulado al real y, posteriormente en el desarrollo gráfico, se trata el tema de la distribución y apariencia.

Ilustración 16 - Jerarquía de clases: servidor



El servidor se desarrolla para trabajar independientemente del cliente (en operaciones como watchers). Por el contrario, el cliente requiere del servidor para realizar cualquier petición. Esto se debe a que el objetivo es que el servidor funcione constantemente recibiendo peticiones y ejecutando watchers; y el cliente solo se use cuando se quiera configurar.

En el cliente `BienvenidaFrame.Java` es la clase que ejecuta el frame donde se colocarán todas las demás vistas, además de ser el intermediario de comunicación entre módulos y entre módulos y el `Cliente.Java` (que es el encargado de realizar peticiones y recibir las respuestas). De esta forma en `BienvenidaFrame.Java` se crea una instancia de cada módulo y en cada módulo existe un puntero hacia `BienvenidaFrame.Java` para acceder a él o a los módulos. Cada módulo guarda datos relativos a su funcionalidad.

- `Cliente.Java`: códigos de error, códigos de petición, timeout e IP y puerto del cliente.
- `Conexión.Java`: IP y puerto del servidor, estado (conectado/desconectado).
- `Configuracion.Java`: IP's disponibles del servidor (eliminando broadcast y localhost).
- `Enrutamiento.Java`: tabla completa de enrutamiento.

- `Watchers.Java`: watchers creados en el servidor.

Los mensajes mediante el protocolo UDP se limitan a 64KB. Se deja abierta la posibilidad de ampliación de ese máximo se amplíe en un futuro con nuevas funcionalidades. Por el momento, la tabla de enrutamiento es la respuesta que más espacio ocupa en los mensajes. Una tabla de 20 tuplas son aproximadamente 3KB. Por lo tanto, 64KB serían un total de 427 tuplas.

El puerto del cliente se selecciona aleatoriamente entre los disponibles. Dar elección al usuario sobre el puerto en la máquina del cliente posee un impacto irrelevante. Sin embargo, sí es importante dar esa posibilidad en el servidor. Aplicar cambios al servidor que modifiquen la IP o el puerto hará que el cliente se reconecte de forma automática.

El software no se ocupa de establecer las rutas en las tablas de enrutamiento para que cliente y servidor se conecten. Han de estar, de antemano, configuradas las tablas para que la máquina que alberga el cliente llegue a la máquina que alberga el servidor. Como ejemplo, podemos hacer una petición para borrar una línea de enrutamiento que contenga la ruta desde el servidor al cliente y perderíamos de inmediato la conexión tras lanzar la petición (su poniendo que se realiza con éxito).

Los watchers se han establecido con un refresco de 8 segundos y éste no puede ser modificado (esta medida evita el consumo excesivo de recursos), aunque no hay un máximo de watchers simultáneos establecido. Además, se ejecutan en segundo plano para que no interfieran con la función principal del servidor: recibir peticiones.

Existen filtros para casi todos los campos de texto modificable. Existen tanto de tipo (String, int, boolean…), como de contenido (que sea una IP, puerto, timeout, interfaz…). Adicionalmente, se reportan, la mayoría de las veces, estos *missmatch* mediante pop-up's con información sobre el mismo.

El servidor y el cliente crean un archivo txt en el mismo lugar de instalación: `IPr.txt` para cliente; `Wr.txt` para el servidor. `IPr.txt` guarda las 10 conexiones exitosas más recientes de IP y puerto. `Wr.txt` guarda todos los watchers que se crearon en el servidor la última vez que se ejecutó. Ambos archivos pueden ser modificados manualmente, pero están pensados para ser creados y modificarlos tanto por el cliente como por el servidor y en una ejecución normal este aspecto debería ser totalmente transparente para el usuario.

En total el código fuente, contando la parte de interfaz gráfica también, consiste en:

- 2 Proyectos
- 13 clases
- 4 iconos
- 4210 líneas de código

5.3 DESARROLLO: INTERFAZ GRÁFICA

La interfaz gráfica se desarrolla siguiendo las pautas de diseño. En el diseño se especifica la distribución de los elementos y en esta parte se les da color, forma y estética. La implementación de la interfaz es una tarea trivial, por lo que nos centraremos en la justificación del aspecto visual.

Se habla siempre de la interfaz gráfica del cliente, ya que el servidor no posee una como bien se ha explicado anteriormente.

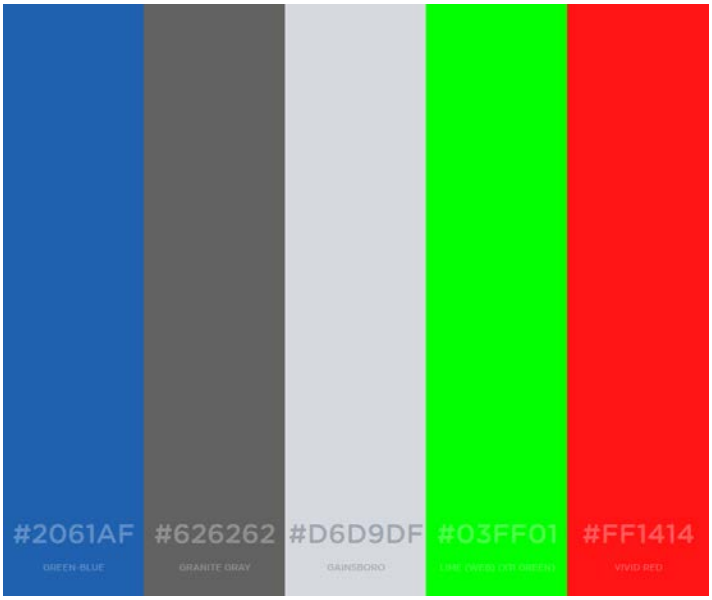
5.3.1 Apariencia

Se define un tamaño de ventana de `850x680p`, sin posibilidad de modificación. El tamaño es suficiente para: a) verse de una forma clara en un mínimo de 11" y resolución 720p[25]; b) distribuir los elementos gráficos especificados en el diseño.

Ilustración 17 - INTA logo

Dado que el software está siendo, concretamente, desarrollado para la empresa INTA. Utilizaremos su logo para extraer el azul y el gris que los utilizaremos como fondo. Añadiremos un gris más oscuro para resaltar y un verde y rojo para estados. Esos cinco colores serán los principales y compondrán la paleta de color.

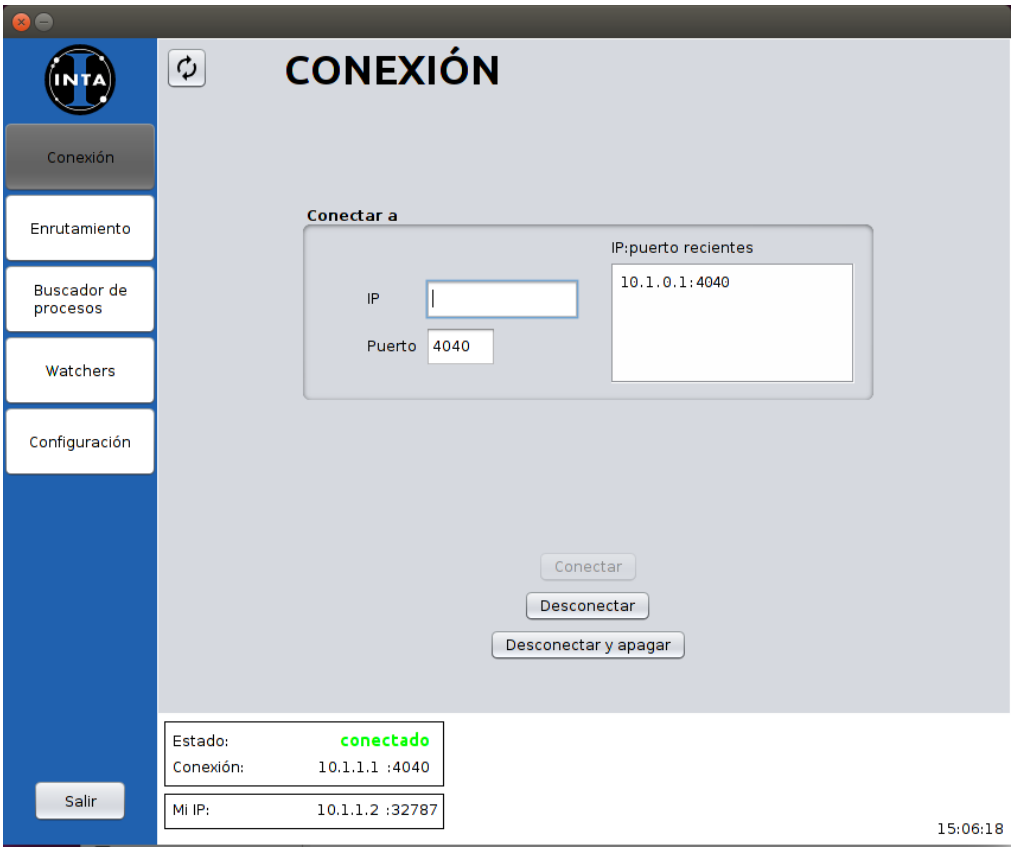
Ilustración 18 - Paleta de colores



[25] Actualmente, casi todos los monitores para ordenadores superan estas características.

Perseguimos una estética consistente en todas las vistas. Se va a analizar la primera vista en profundidad y las demás, simplemente, se detallarán diferencias únicas con las otras.

Ilustración 19 - Vista: conexión



Las tres zonas del diseño están claramente divididas por colores: 1) Menú; 2) Zona del módulo; 3) Zona de información.

Observamos, el menú a la izquierda claramente diferenciado con el color azul. Con gris más oscuro se muestra en qué vista estamos y, además, el nombre coincide con el título de la vista.

En la Zona del módulo, os botones de "Conectar", "Desconectar" y "Desconectar y apagar" se diferencian cuales están habilitados y cuáles no mediante una transparencia mayor. El recuadro que separa los elementos de "conectar a" hace un efecto hundido para destacarlo.

En la zona de información, los estados se representan por colores: a) verde = conectado; b) rojo = desconectado; c) naranja = procesando (aunque este estado nunca debería de aparecer). Las IP se cuadran en una misma columna para facilitar su lectura.

Ilustración 20 - Z.I.: desconectado

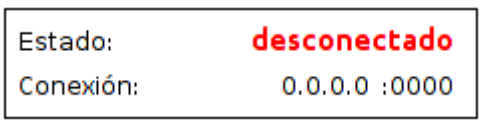
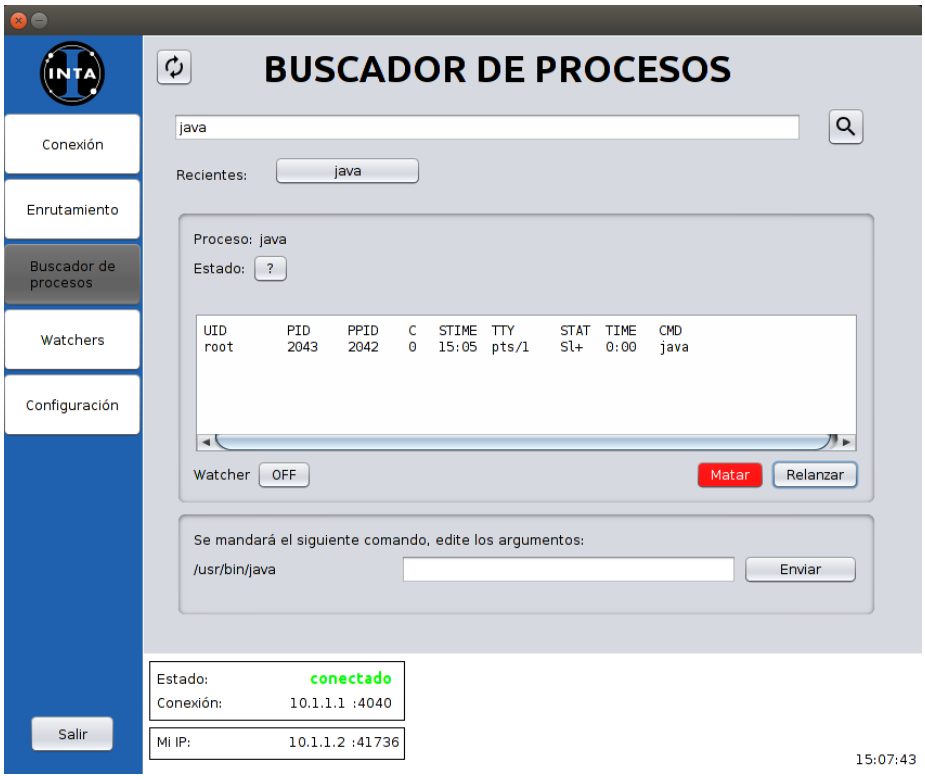


Ilustración 21 - Vista: enrutamiento

2 nuevos elementos gráficos en la Zona del módulo: una tabla y un desplegable. La tabla distingue su título del contenido por el uso de un recuadro y el uso de MAYÚSCULAS. Además, separa, sustancialmente, el botón de la *papelera* para evitar acciones no deseadas.

Ilustración 22 - Vista: Buscador de procesos



Utilizamos el color rojo como señal de que es una acción con graves consecuencias. Tiene para alertar, no para destacar.

Ilustración 23 - Vista: watchers

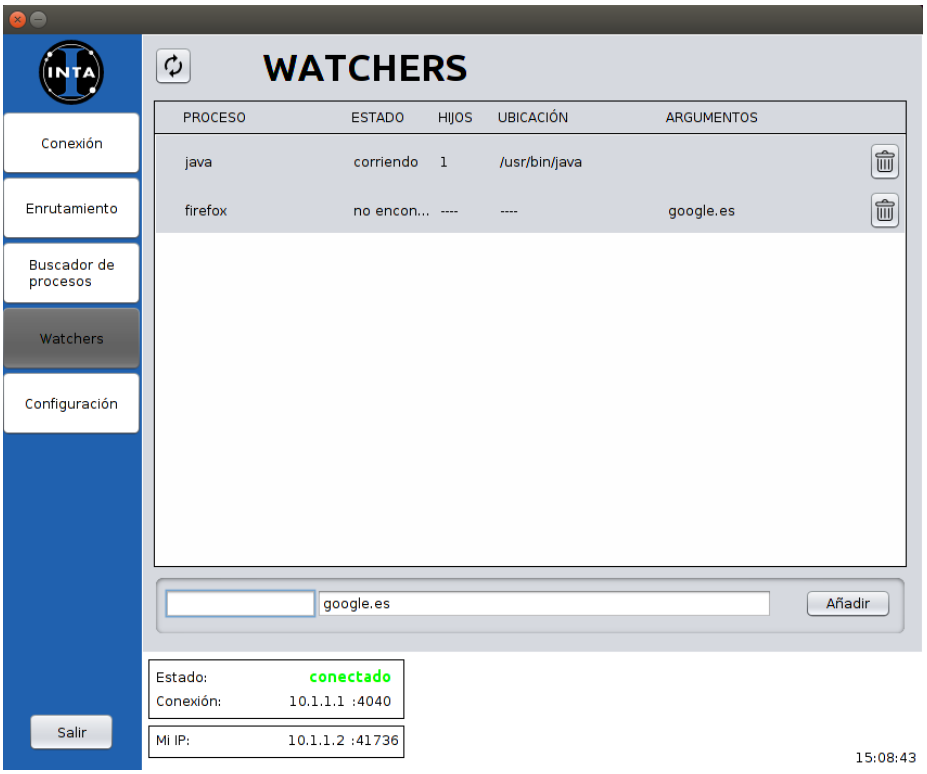


Ilustración 24 - Vista: configuración

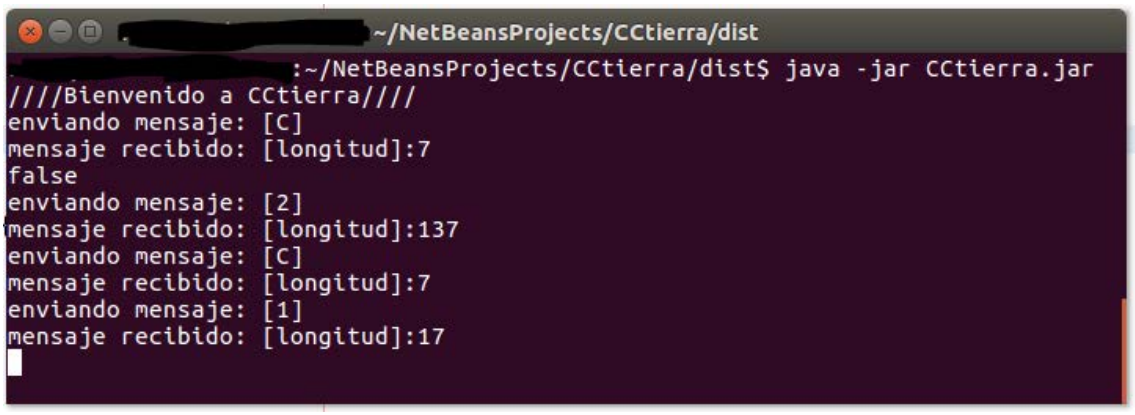
En ninguna de las anteriores vistas se han implementado animaciones de transición: consumen recursos y no aporta diferencia a los objetivos que se pretenden conseguir.

5.3.2 Feedback

En cada vista existe un botón de refresco (icono: ) ya que, de forma periódica, las vistas no se refrescan automáticamente. Esto es debido a que existe un riesgo de saturar las peticiones.

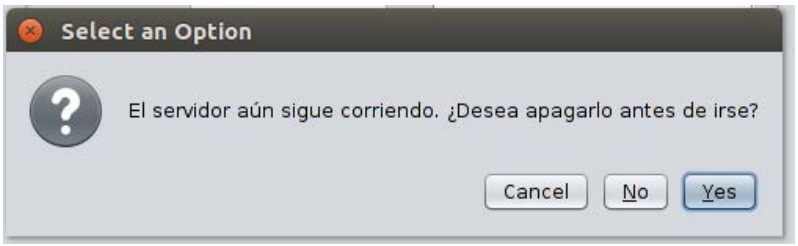
Existe una diferencia sustancial de las vistas del desarrollo con los wireframes del diseño: se elimina el feedback de errores con la zona de información. Realmente no se elimina si no que se redirige la información de los errores a la salida del terminal, tanto en el servidor como en el cliente. Debido a que se persigue la usabilidad, este elemento gráfico influye de forma negativa. Por lo tanto, debe colocarse en un segundo plano como es la salida del terminal, que en una ejecución normal es transparente para el usuario.

Ilustración 25 - Salida del terminal



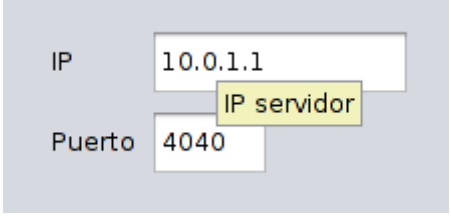
Adicionalmente, se ofrece una versión reducida de la información de los errores en forma de pop-up.

Ilustración 26 - Salida Pop-up



Por último, se añaden en la mayoría de elementos gráficos un texto que aparece cuando pasas el ratón por encima. El texto identifica con un nombre auto explicativo el elemento o aclara en caso de ambigüedad.

Ilustración 27 - Hover



6 EVALUACIÓN Y PRUEBAS

En este apartado se detallan las pruebas realizadas a los dos procesos de desarrollo llevador a cabo: nivel interno e interfaz gráfica: a nivel interno se evalúa con el programa; y la interfaz gráfica se evalúa con la ayuda de un tercero.

Es necesario destacar la importancia de pruebas exhaustivas debido a la decisión de diseño de programar un sistema robusto a errores por estar alojado, el servidor, en un dispositivo de suma importancia.

6.1 PRUEBAS: NIVEL INTERNO

Las pruebas del nivel interno se realizan sobre una ejecución normal, es decir, sobre GSC e IDAN. Se realiza una tarea y se espera una respuesta. Si la respuesta real es la respuesta esperada, la prueba ha sido superada con éxito. Al ser numerosas pruebas, se ofrece un detalle de cada prueba en el Anexo I. Hubo una primera tanda de pruebas (#1) al terminar el desarrollo del nivel interno. Hubo errores por lo que se aprovecha la etapa del desarrollo de la interfaz gráfica para rediseñar y parchear. En la segunda tanda de pruebas (#2) se realiza el mismo día que las pruebas de la interfaz gráfica. A continuación, se ofrece una tabla reducida con la información de las pruebas del nivel interno:

Tabla 9 - Pruebas: nivel interno

Nombre simbólico	#1	#2
Servidor ip	✓	✓
Servidor puerto	✓	✓
Servidor timeout	✓	✓
Servidor sin argumentos	✓	✓
Servidor error arg.	✓	✓
Conexión ip y puerto	✓	✓
Conexión ip no válida	✓	✓
Conexión puerto no válido	✓	✓
Desconexión	✓	✓
Buscar proceso existente	✓	✓
Buscar proceso inexistente	✓	✓
Matar proceso existente	✓	✓
Matar proceso inexistente	✓	✓
Relanzar proceso	✗	✓
Eliminar línea enrutamiento	✓	✓
Añadir línea enrutamiento	✗	✓
Añadir línea enrutamiento no válida	✓	✓
Crear watcher	✗	✓
Crear watcher no válido	✓	✓
Eliminar watcher	✓	✓
Modificar ip servidor	✗	✓
Modificar puerto servidor	✓	✓
Modificar timeout servidor	✓	✓

Modificar timeout cliente	✓	✓
Forzar error 01[26]	✓	✓
Forzar error 02	✓	✓
Forzar error 03	✓	✓
Forzar error 04	✓	✓
Forzar error 05	✓	✓
Forzar error 06	✓	✓
Forzar error 07	✓	✓
Forzar error 08	✓	✓
Forzar error 09	✓	✓
Forzar error 0A	✓	✓
Forzar error 0B	✓	✓
Forzar error 1B	✓	✓
Forzar error 0C	✓	✓
Forzar error 0D	✓	✓
Forzar error 0E	✓	✓
Forzar error 0F	✓	✓
Forzar error 0G	✓	✓
Forzar error 0H	✓	✓
Forzar error 99	✗	✓

Como vemos en la segunda tanda de pruebas se han superado todas con éxito. Todas las funcionalidades del diseño a nivel interno quedan implementadas de forma satisfactoria.

6.2 PRUEBAS: INTERFAZ GRÁFICA

Para las pruebas de interfaz gráfica se cuenta con la colaboración de un tercero. Él es un usuario habitual del UAV con amplia experiencia en utilización del mismo; además, ofreció ayuda a la hora de especificar los requisitos. Por lo tanto, será uno de los usuarios habituales en utilizar el software desarrollado, ergo es un candidato ideal para la prueba. Dada la ocupación de los otros miembros, solo vamos a disponer de esta prueba con esta persona para concluir aspectos positivos y negativos sobre la interfaz.

La primera parte de la prueba consiste en medir el tiempo de la realización de tareas. Él evaluado nunca antes ha visto el software, por lo que obtenemos unas primeras impresiones de la usabilidad. A continuación, se ofrece una tabla con los resultados que realizó frente a una persona experta en el sistema desarrollado.

Tabla 10 - Prueba interfaz gráfica

Tarea	Víctor	Usuario experto
Instalar software	125s	30s
Ejecutar software	15s	14s
Conectar con el servidor	16s	4s
Añadir ruta 10.0.2.0/24	10s	6s
Eliminar ruta añadida	4s	2s

[26] Como son errores que se notifican en situaciones concretas, tenemos que forzarlos para que ocurran. Por lo tanto, se modifica el servidor para que devuelva el código de error aun habiendo sido satisfecha la operación. El propósito es comprobar que se notifican adecuadamente los errores.

Cambiar IP, puerto y timeout del servidor	16s	8s
Buscar el proceso 'ping'	7s	4s
Relanzar el proceso 'ping'	8s	4s
Añadir watcher sobre 'Java' sin argumentos	15s	6s
Añadir watcher sobre 'ping' con '10.1.1.2'	15s	6s
Eliminar watcher sobre 'ping'	5s	2s
Matar proceso 'ping'	6s	6s
Apagar el servidor	6s	4s
Cerrar el cliente	4s	1s

Se pregunta el motivo en los tiempos que han tardado más de diez segundos o el doble con respecto al del usuario experto. Durante la instalación Él asegura que ha tardado porque no se conocía los comandos para conectar mediante ssh y para mandar archivos por scp, no porque las instrucciones no hayan quedado claras. Dice que tardó en conectar al servidor porque estaba identificando todos los elementos gráficos por primera vez y después de haberlos identificados las siguientes tareas fueron sencillas. En la tarea de añadir los watchers no le quedaba claro para qué servían y buscó la información en ambas ocasiones.

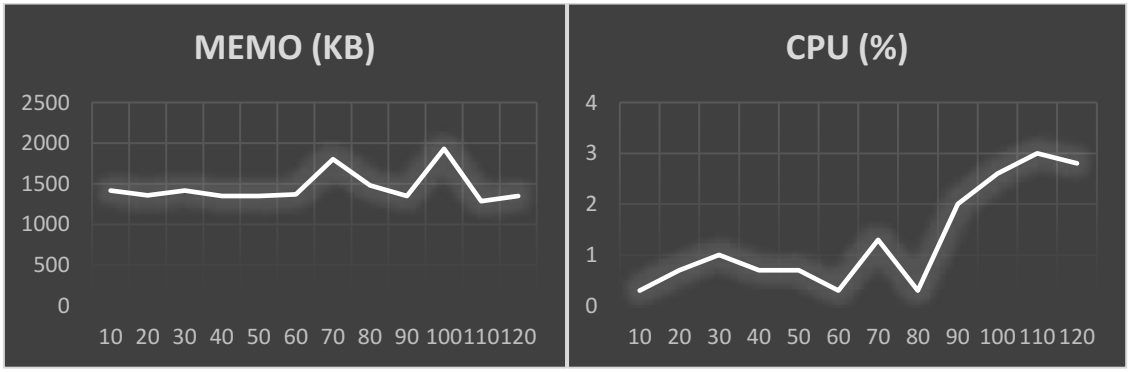
Adicionalmente, en una segunda parte, se le deja utilizar el programa sin restricciones y dar su opinión mientras lo hace. Añadió que, de forma general, estaba satisfecho con la funcionalidad y por cómo se veía. El único aspecto negativo que destacó es la ejecución desde el terminal, proponiendo como solución una aplicación de escritorio ejecutable. Después de un par de pasadas por todas las vistas ya se comportaba como un usuario experto.

Concluimos que es una indicación de que se han cumplido los objetivos del diseño y de los requisitos de la interfaz gráfica.

6.3 CONSUMO DE RECURSOS

Por último, se miden los recursos realizando operaciones variadas (en el primer experimento elegidas aleatoriamente entre todas las posibles y replicando ese primer experimento 9 veces más) y midiendo el rendimiento medio cada segundo. A continuación, se expone un gráfico con los resultados en intervalos de 10 segundos durante 120 segundos. Los resultados son la media de 10 experimentos utilizando el mismo ciclo de operaciones.

Gráfico 1 - Consumo memoria y CPU



Al final de la medición, se crean un total de 20 watchers y comprobamos que está directamente relacionado con el rendimiento de la CPU. Sin embargo, aun llevando el sistema a extremos (con 50 watchers y realizando operaciones de petición de información constantemente),

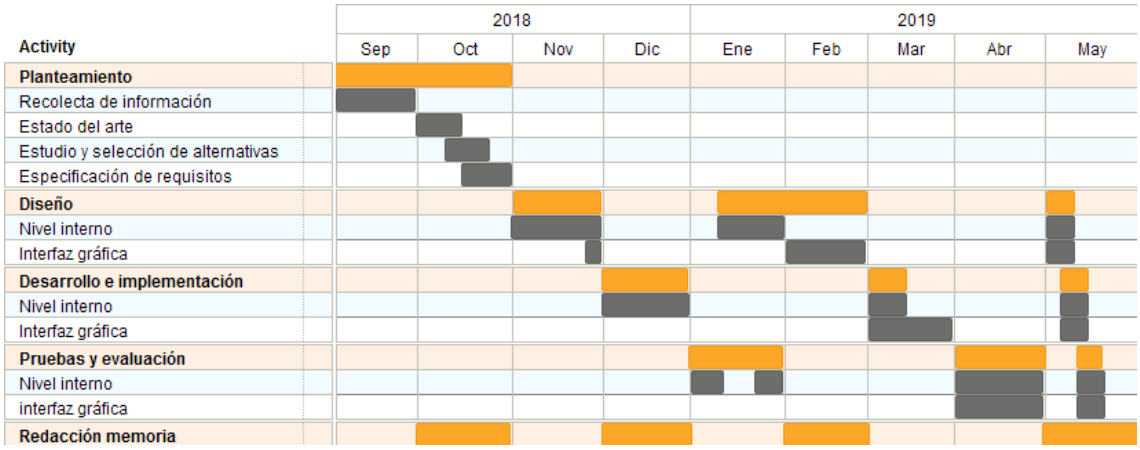
no ocupa más de 2MB de memoria y no consume más de un 3.5% de CPU. Ambas son cifras más que aceptables.

Concluimos que se alcanza el objetivo de eficiencia en consumo de recursos.

7 PLANIFICACIÓN Y RECURSOS

Al comienzo del proyecto, en septiembre, se planifica por meses cómo tienen que cumplirse las diferentes etapas del proyecto teniendo en cuenta el horizonte temporal de junio. En total, el plazo es de 9 meses. He aquí la planificación:

Tabla 11 - Diagrama Gantt



Todos los objetivos de cada mes se han cumplido en el plazo especificado. No ha sido posible paralelizar casi ninguna tarea debido a que el proyecto lo realiza, en su totalidad, una sola persona. Sin embargo, sí existía la compatibilidad de algunas tareas para ser totalmente paralelizadas en caso del que proyecto hubiera tenido más participantes.

Tabla 12 - Recursos

RECURSOS MONETARIOS		
Concepto		Valoración
Equipo **-Máquina UAV x2** **-Máquina router** **-Periféricos(monitor, ratón/teclado, Switch y USB)**	 6000€ x 2 uds. 300€ 150€	 12.450 €
Desplazamiento[27] (Abono transporte mensual)	20€ * 7 meses	140 €
Laboratorio (luz, agua, internet…)	75€ * 7 meses	525 €
Papelería (papel, bolígrafos, tinta…)		42 €
	Total monetario:	**13.157 €**
RECURSOS TEMPORALES		
Concepto	Tiempo	Valoración
Horas hombre invertidas en el proyecto por el autor	440 h * 12,43 [28]€/h	~5.469 €
Reuniones	15 h * 27[29]€/h	405 €
Pruebas con terceros	3 h * 24.86[30]€/h	~75 €
	Total temporales:	**5949 €**
	TOTAL FINAL:	**19.106 €**

[27] Los meses de octubre, febrero y mayo no es necesario el desplazamiento
[28] Estimación entre el salario medio de un desarrollador informático (véase: (2019)-2) y un ingeniero de software (véase: Indeed (2019)-1) en España.
[29] Suma del salario medio por hora del apartado anterior con un profesor en España. Véase: Indeed (2019)-3
[30] Suma de dos salarios mediso de un desarrollador informático (véase: (2019)-2) y un ingeniero de software (véase: Indeed (2019)-1) en España.

Sin duda alguna, las tareas más críticas fueron: a) la especificación de requisitos; b) diseño del nivel interno y de la interfaz gráfica. Estos dos aspectos repercuten en tareas posteriores y además un error en ellos acarrea un error incorregible, a no ser que se rediseñe toda esa parte; suponiendo una pérdida te tiempo y recursos considerable.

En este proyecto ha habido dos tipos de recursos: monetario y temporal. El monetario es fácilmente cuantificable, pero el temporal ha de ser valorado subjetivamente.

El coste final del proyecto serían 19.106€, aunque en realidad habría que descontar el coste de las máquinas UAV porque ya estaban disponibles previamente al proyecto. Sin embargo, se decide incluirlas para dar una visión general y de proyecto que parte de cero.

8 CONCLUSIONES Y TRABAJO FUTURO

Este ha sido un proyecto exigente en el que se ha tenido que desarrollar por completo un software para un sistema crítico. Comprender las necesidades del usuario, un diseño detallado y una buena planificación han sido los puntos más importantes del trabajo, siendo, ellos, la clave del éxito o fracaso de las demás tareas y del producto final. Esos tres puntos realizados correctamente ahorran tiempo y recursos que se pueden invertir en la mejora del propio software.

El cliente final ha quedado satisfecho con el producto resultante de este trabajo y declara que le será de utilidad para realizar operaciones de configuración, además de que otros usuarios (no expertos) podrán realizar las mismas operaciones sin su supervisión. Por lo tanto, concluimos que se ha alcanzado una solución al problema planteado y el proyecto ha sido un éxito.

El código fuente y las versiones se pueden encontrar en:

https://github.com/Romeorubiko/CCTierra

https://github.com/Romeorubiko/CCAire

8.1 POSIBLES MEJORAS DEL SISTEMA EN UN FUTURO

Dada la característica de escalabilidad es necesario sugerir posibles trabajos futuros para la expansión o mejora del software. Cuando hablamos de expansión, nos referimos a escalarlos a otras plataformas o cambiar su arquitectura considerablemente; cuando hablamos de mejora del software, nos referimos añadir, sin mucha complejidad, nuevas funcionalidades sin necesidad de expandir el sistema. A continuación, se listan algunas ideas para funcionalidades:

- Sustituir la función de Wathers y utilizar un software de terceros especializado para ello como puede ser: monit, supervisor o daemontools.
- Añadir la funcionalidad de configurar servicios de la máquina alojada en el dron.
- Ser posible iniciar le servidor desde la interfaz gráfica del cliente.
- Añadir la funcionalidad de operaciones programadas para una cierta hora.
- Posibilidad de guardar la configuración en una nube.
- Añadir un paso de identificación con usuario y clave.

Eso en cuanto a funcionalidades. El software podrá expandirse de las siguientes maneras:

- Aplicación Windows
- Aplicación Android/iOS/Tizen
- Aplicación web
- Soporte para plugins
- Espacio para subir, descargar y configurar plugins
- Posibilidad de determinar el protocolo de transmisión (TCP/UDP)

Como no se puede prever el futuro, se recomienda siempre cubrir las necesidades del cliente: será, en gran medida, quién determine el camino del software.

9 BIBLIOGRAFÍA

Acuña, Jose María (2017) - Características y tabla comparativa de los sistemas operativos móviles más usados. Recuperado de: https://jmacuna.tecnoblog.guru/2017/03/sistemas-operativos-moviles.html

Ansible (2019) – Get started. Ready to start automating?. Recuperado de: https://www.ansible.com/resources/get-started

Aliprandi, Simone (2011) - Creative Commons: a user guide.

Bermstein, F.J. (2019) – daemontools. Recuperado de: https://cr.yp.to/daemontools.html

Beth Rosson, Mary. M.Carroll, John (2002). Usability Engineering

Cerf, Cinton. Dalal, Yogen. Sunshine, Carl. (1974) – Specification of the Internet Transmision Control Program.

Free Software Foundation, Inc. (2007) - GNU General Public License. Versión 3. Recuperado de: http://www.gnu.org/copyleft/gpl.html

IANA (2019) - Service Name and Transport Protocol Port Number Registry, pp. 77. Recuperado de: https://www.iana.org/assignments/service-names-port-numbers/service-names-port-numbers.xhtml

IEEE (2008) - Especificación de Requisitos según el estándar de IEEE 830. IEEE Std. 830-1998

Indeed (2019)-1 - Salarios para empleos de Ingeniero/a de software en España. Recuperado de: https://www.indeed.es/salaries/Ingeniero/a-de-software-Salaries?period=monthly

Indeed (2019)-2 - Salarios para empleos de Desarrollador/a de software en España. Recuperado de: https://www.indeed.es/salaries/Desarrollador/a-de-software-Salaries?period=monthly

Indeed (2019)-3. Salarios para empleos de Profesor/a en España. Recuperado de: https://www.indeed.es/salaries/Profesor/a-Salaries

UC3M Laboratorio DEI (2015) - Interfaces de usuario curso 2015/2016.

ISO (2018) - ISO 9241-11:2018. Ergonomics of human-system interaction -- Part 11: Usability: Definitions and concepts.

Lehtinen, S. (2006) - The Secure Shell (SSH) Protocol Assigned Numbers.

Monit (2019) – Monit manual. Versión: 5.25.3. Recuperado de https://mmonit.com/monit/documentation/monit.html

Postel, J. (1980) – User Datagram Protocol.

Preece, J., Rogers, Y., & Sharp, H. (2002). Interaction Design: Beyond Human-Computer Interaction. New York: John Wiley & Sons.

RTD (2019) - IDAN® Building Block & System Selection. Recuperado de: https://www.rtd.com/IDAN/IDAN_selection.htm

Supervisor (2019) – Supervisor manual. Versión: 4.0.2. Recuperado de: http://supervisord.org/introduction.html

9.1 RECURSOS ELECTRÓNICOS

Freeflight (2019):

- Android: https://play.google.com/store/apps/details?id=com.parrot.freeflight&hl=es
- iOS: https://itunes.apple.com/us/app/freeflight/id373065271?mt=8

Pix4Dcapture (2019):

- Android: https://play.google.com/store/apps/details?id=com.pix4d.pix4dmapper
- iOS: https://itunes.apple.com/us/app/pix4dcapture/id953486050?ls=1&mt=8

Mission Planner (2019):

- Windows: http://ardupilot.org/planner/index.html

Hover (2019):

- Android: https://play.google.com/store/apps/details?id=com.analyticadevelopment.android.hover&hl=es
- iOS: https://itunes.apple.com/us/app/hover-1-app-for-drone-dji-freeflight-pilots/id947641516?mt=8

ANEXO I – BATERÍA DE PRUEBAS DETALLADA

Nombre simbólico	Descripción	Resultado esperado
Servidor ip	Se ejecuta el servidor en una ip introducida por argumentos	El servidor se inicia con normalidad
Servidor puerto	Se ejecuta el servidor en una ip y un puerto introducidos por argumentos	El servidor se inicia con normalidad
Servidor timeout	Se ejecuta el servidor en una ip y con un timeout introducidos por argumentos	El servidor se inicia con normalidad
Servidor sin argumentos	Se ejecuta el servidor sin argumentos	El servidor imprime las ip's disponibles y los argumentos aceptados
Servidor error arg.	Se ejecuta el servidor con argumentos erróneos, es decir, ni de ip, ni de puerto, ni de timeout	El servidor imprime un error de que desconoce los argumentos
Conexión ip y puerto	Estando el servidor iniciado, conectamos con el cliente a su ip y puerto	Conexión entre cliente y servidor satisfactoria
Conexión ip no válida	Conectamos el cliente a una ip donde no está el servidor	Pop-up en el cliente informando del error
Conexión puerto no válido	Conectamos el cliente a una ip donde se encuentra el servidor, pero en un puerto distinto	Pop-up en el cliente informando del error
Desconexión	Estando el servidor conectado al cliente, desconectamos el cliente del servidor	El servidor se apaga y el cliente muestra desconexión
Buscar proceso existente	Estando conectado al servidor, en el cliente buscamos un proceso que esté corriendo en el servidor	El cliente muestra información sobre el proceso
Buscar proceso inexistente	Estando conectado al servidor, en el cliente buscamos un proceso que esté en el servidor	El cliente muestra información sobre su inexistencia
Matar proceso existente	Estando conectado al servidor, en el cliente buscamos un proceso que esté corriendo en el servidor y lo matamos	El proceso del servidor muere y en el cliente se muestra información al respecto
Matar proceso inexistente	Estando conectado al servidor, en el cliente buscamos un proceso que no esté corriendo en el servidor y lo matamos	El botón de 'matar' está deshabilitado.
Relanzar proceso	Estando conectado al servidor, en el cliente buscamos un proceso que no esté corriendo en el servidor y que exista y lo relanzamos	El proceso del servidor se ejecuta y en el cliente se muestra información al respecto
Eliminar línea enrutamiento	Estando conectado al servidor, en el cliente borramos una línea de la tabla de enrutamiento	El servidor elimina esa línea de su tabla de enrutamiento y en el cliente se muestra información al respecto
Añadir línea enrutamiento	Estando conectado al servidor, en el cliente añadimos una línea de la tabla de enrutamiento	El servidor añade esa línea de su tabla de enrutamiento y en el cliente se muestra información al respecto

Añadir línea enrutamiento no válida	Estando conectado al servidor, en el cliente añadimos una línea de la tabla de enrutamiento no válida.	El servidor devolverá un error 05 que será notificado por el cliente al usuario
Crear watcher	Estando el cliente conectado al servidor, mandamos la petición de crear un watcher	El servidor crea el watcher y se ve reflejado en el cliente
Crear watcher no válido	Estando el cliente conectado al servidor, mandamos la petición de crear un watcher siendo este no válido	El servidor crea el watcher y se ve reflejado en el cliente como no válido
Eliminar watcher	Estando el cliente conectado al servidor, mandamos la petición de eliminar un watcher	El servidor elimina el watcher y se ve reflejado en el cliente
Modificar ip servidor	Estando conectado al servidor, en el cliente modificamos la ip del servidor	El servidor cambia de ip y el cliente se reconecta automáticamente a esa nueva ip
Modificar puerto servidor	Estando conectado al servidor, en el cliente modificamos el puerto del servidor	El servidor cambia de puerto y el cliente se reconecta automáticamente a ese nuevo puerto
Modificar timeout servidor	Estando conectado al servidor, en el cliente modificamos el timeout del servidor	El servidor cambia de timeout.
Modificar timeout cliente	En el cliente modificamos el timeout del cliente	El cliente cambia de timeout
Forzar error 01[31]	Se modifica el servidor para que devuelva el error.	Se notifica de un error 01
Forzar error 02	Se modifica el servidor para que devuelva el error.	Se notifica de un error 02
Forzar error 03	Se modifica el servidor para que devuelva el error.	Se notifica de un error 03
Forzar error 04	Se modifica el servidor para que devuelva el error.	Se notifica de un error 04
Forzar error 05	Se modifica el servidor para que devuelva el error.	Se notifica de un error 05
Forzar error 06	Se modifica el servidor para que devuelva el error.	Se notifica de un error 06
Forzar error 07	Se modifica el servidor para que devuelva el error.	Se notifica de un error 07
Forzar error 08	Se modifica el servidor para que devuelva el error.	Se notifica de un error 08
Forzar error 09	Se modifica el servidor para que devuelva el error.	Se notifica de un error 09
Forzar error 0A	Se modifica el servidor para que devuelva el error.	Se notifica de un error 0A
Forzar error 0B	Se modifica el servidor para que devuelva el error.	Se notifica de un error 0B
Forzar error 1B	Se modifica el servidor para que devuelva el error.	Se notifica de un error 1B
Forzar error 0C	Se modifica el servidor para que devuelva el error.	Se notifica de un error 0C
Forzar error 0D	Se modifica el servidor para que devuelva el error.	Se notifica de un error 0D
Forzar error 0E	Se modifica el servidor para que devuelva el error.	Se notifica de un error 0E
Forzar error 0F	Se modifica el servidor para que devuelva el error.	Se notifica de un error 0F
Forzar error 0G	Se modifica el servidor para que devuelva el error.	Se notifica de un error 0G
Forzar error 0H	Se modifica el servidor para que devuelva el error.	Se notifica de un error 0H
Forzar error 99	Se modifica el servidor para que devuelva el error.	Se notifica de un error 99

[31] Como son errores que se notifican en situaciones concretas, tenemos que forzarlos para que ocurran. Por lo tanto, se modifica el servidor para que devuelva el código de error aun habiendo sido satisfecha la operación. El propósito es comprobar que se notifican adecuadamente los errores y se recupera el sistema después de haber sufrido uno.

ANEXO II – DESGLOSE PLANIFICACIÓN

MES	TAREA
Septiembre	**-Planteamiento** del proyecto -Recolecta de información sobre las **funcionalidades y limitaciones** del programa.
Octubre	-Especificación de **requisitos y limitaciones** del programa. **-Decisiones de diseño**.
Noviembre	**-Diseño** a nivel interno. **-Diseño** de una interfaz básica.
Diciembre	**-**Creación del **entorno de trabajo**. **-Desarrollo** a nivel interno.
Enero	-Primera batería de **pruebas** **-Corrección de errores** a nivel interno **-Re-Diseño** a nivel interno
Febrero	**-Diseño** de interfaz
Marzo	**-Re-Desarrollo** a nivel interno **-Desarrollo** de interfaz gráfica
Abril	-Segunda batería de **pruebas** **-Prueba** interfaz gráfica
Mayo	**-Redacción** memoria TFG **-Últimos retoques** de errores(software y memoria)
Junio	**-Entrega** del TFG **-Presentación** del TFG